ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/350838466

Polycopié de cours et des travaux pratiques. Matière : Programmation JAVA

Technical Report · April 2021

DOI: 10.13140/RG.2.2.36836.30089

CITATIONS
0

READS
3,708

1 author:

M'hamed Bilal Abidine
University of Science and Technology Houari Boumediene (USTHB)

59 PUBLICATIONS **332** CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Project Human Activity Recognition in Smart Home based on Wireless Sensor Networks View project

Project Human Activity Recognition Using Smartphones View project

All content following this page was uploaded by M'hamed Bilal Abidine on 13 April 2021.

The user has requested enhancement of the downloaded file.

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université des Sciences et de la Technologie Houari Boumedienne
FACULTE D'ELECTRONIQUE ET D'INFORMATIQUE
Département Télécommunications

Polycopié des Travaux Pratiques

Matière : Java

Auteur : ABIDINE M'hamed Bilal

Maitre de Conférences Classe «B»

2019

Organisation générale de l'ouvrage :

Objectif……3
Introduction à la programmation orientée objet…… **4**
TP 0 : Initiation à la programmation Java avec Eclipse**……6**
TP 1 : Classes et Objets Java avec Eclipse…… **18**
TP 2 : Méthodes d'accès aux attributs: Getters et Setters …… **23**
TP 3 : Tableaux et vecteurs …… **28**
TP 4 : Héritage et Polymorphisme **……**…… **32**
TP 5 : Interface Graphique et Animation …… **38**
Solution TP0…… 42
Solution TP1…… 45
Solution TP2…… 49
Solution TP3…… 55
Solution TP4…… 60
Solution TP5…… 65

Objectif :

L'objectif de cet ouvrage est d'apprendre à l'étudiant de concevoir des programmes valides et fiables avec la plate-forme de développement Eclipse de Java. Le présent polycopié est un support pédagogique qui rassemble une panoplie d'exercices des travaux pratiques (TP) élaborés et dispensés à l'USTHB pour les étudiants de la première année Master : Communications Radio Mobile (C.R.M). Ce sont des TP conçus spécialement pour des étudiants en Télécommunications qui suivent un programme de cours divers et varié en la matière. Plusieurs exercices ont été proposés dans cet ouvrage tout en donnant les notions fondamentales à l'étudiant pour une compréhension simple et rapide. Le niveau d'exercices augmentait d'un TP à l'autre afin de rendre l'étudiant autonome dans l'implémentation des programmes.

Le langage Java est un langage de programmation orienté objet créé en 1995 par Sun Microsystems. Il permet une programmation orientée-objet et modulaire - on peut écrire des portions de code génériques, c'est-à-dire utilisables par plusieurs applications. Il est dit aussi portable puisque un même programme peut être compilé et exécuté sur différents environnements. Pour l'implémentation des programmes, l'étudiant devra choisir la version Eclipse Luna, une plate-forme ouverte pour le développement d'applications et extensible grâce à un mécanisme de plug-ins.

Nous verrons dans ce polycopié les caractéristiques générales et spécifiques du langage orienté objet Java : notions de classe, d'attribut, de méthode, d'instance (objet), de constructeur, de destructeur, Tableaux, chaînes de caractères, types primitifs d'interface, de paquetage, d'héritage, polymorphisme, d'encapsulation, redéfinition de méthodes, interfaces utilisateur à l'aide de "swing", et application de cette approche à la réalisation de jeux vidéo.

Introduction à la programmation orientée objet :

La programmation orientée objet est l'une des méthodologies récentes de programmation, couramment utilisée par les langages de programmation les plus répandus (C++, Java, C#.Net, ...). Dans cette partie, nous allons définir ce qu'est la programmation orientée objet (abrégée souvent en POO). En fait, on peut dire que la POO est une façon de développer une application qui consiste à représenter (on dit également « **modéliser** ») des éléments d'une partie du monde physique (que l'on appelle *domaine*) sous la forme d'entités informatiques (objets), ayant des propriétés et pouvant interagir entre eux. De plus, les personnes non-techniques pourront comprendre et éventuellement participer à cette modélisation.

Cette façon de modéliser les choses permet également de découper une grosse application, généralement floue, en une multitude d'objets interagissant entre eux. Cela permet de découper un gros problème en plus petits afin de le résoudre plus facilement. La difficulté de cette modélisation consiste à créer une représentation abstraite, sous forme d'objets, d'entités ayant une existence matérielle (chien, voiture, ampoule, ...) ou bien virtuelle (sécurité sociale, temps, ...).

Un objet est caractérisé par plusieurs notions :

- **Les attributs**: Il s'agit des données caractérisant l'objet. Ce sont des variables stockant des informations d'état de l'objet.
- **Les méthodes** (appelées parfois *fonctions membres*): Les méthodes d'un objet caractérisent son comportement, c'est-à-dire l'ensemble des actions (appelées *opérations*) que l'objet est à même de réaliser. Ces opérations permettent de faire réagir l'objet aux sollicitations extérieures (ou d'agir sur les autres objets). De plus, les opérations sont étroitement liées aux attributs, car leurs actions peuvent dépendre des valeurs des attributs, ou bien les modifier
- **L'identité**: L'objet possède une identité, qui permet de le distinguer des autres objets, indépendamment de son état. On construit généralement cette identité grâce à un identifiant découlant naturellement du problème (par exemple un produit pourra être repéré par un code, une voiture par un numéro de série, etc.)

Chaque objet a sa propre vie et est différent d'un autre. Nous pouvons avoir une chaise bleue, une autre rouge, une autre avec des roulettes, une cassée … Il faut bien faire attention à distinguer ce qu'est l'objet et ce qu'est la définition d'un objet.

La définition de l'objet (ou structure de l'objet) permet d'indiquer ce qui compose un objet, c'est-à-dire quelles sont ses propriétés, ses actions etc. Comme par exemple le fait qu'une chaise ait des pieds ou qu'on puisse s'asseoir dessus. Par contre, l'objet chaise est bien concret. On peut donc avoir plusieurs objets chaises : on parle également d'instances. Les objets chaises, ce sont bien celles concrètes que l'on voit devant nous autour de l'objet table pour démarrer une partie de belote.

Un avantage de la POO est l'encapsulation. C'est un mécanisme consistant à rassembler les données et les méthodes au sein d'une structure en cachant l'implémentation de l'objet, c'est-à-dire en empêchant l'accès aux données par un autre moyen que les services proposés. L'encapsulation permet donc de garantir l'intégrité des données contenues dans l'objet.

Un autre avantage de la POO est la réutilisabilité. Des objets peuvent être réutilisés ou même étendus grâce à la notion d'héritage. C'est le cas par exemple de la bibliothèque de classes **swing** que nous allons utiliser dans le dernier TP du manuscrit. Cette bibliothèque nous fournit par exemple tous les objets permettant de construire des applications graphiques. Pas besoin de réinventer toute la mécanique pour gérer des fenêtres dans une application, le **JFrame** avec **swing** sait déjà faire tout ça. Nous avons juste besoin d'utiliser un espace d'exécution appelé **conteneur**, dans lequel nous pourrons mettre un objet « bouton » et un objet « zone de texte ». Ces objets héritent tous des mêmes comportements, comme le fait d'être cliquable ou sélectionnable, etc. De même, des composants tout faits et prêts à l'emploi peuvent être vendus par des entreprises tierces (système de log, contrôles utilisateurs améliorés, etc …).

Il faut savoir que la POO, c'est beaucoup plus que ça et nous en verrons des subtilités plus loin, mais comprendre ce qu'est un objet est globalement suffisant pour une grande partie de ce manuscrit.

Si on prend l'exemple du monde réel, nous sommes entourés d'objets : une chaise, une table, une voiture, etc. Ces objets forment un tout.

- Ils possèdent des propriétés (la chaise possède quatre pieds, elle est de couleur bleue, etc.).
- Ces objets peuvent faire des actions (la voiture peut rouler, klaxonner, etc.).
- Ils peuvent également interagir entre eux (l'objet conducteur démarre la voiture, l'objet moteur fait tourner l'objet voiture, etc.).

En résumé

- L'approche orientée objet permet de modéliser son application sous la forme d'interactions entre objets.
- Les objets ont des propriétés et peuvent faire des actions.
- Ils masquent la complexité d'une implémentation grâce à l'encapsulation.
- Les objets peuvent hériter de fonctionnalités d'autres objets s'il y a une relation d'héritage entre eux.

TP0 : Initiation à la programmation Java avec Eclipse

1. Introduction sur l'IDE Eclipse :

L'IDE (**I**ntegrated **D**evelopment **E**nvironement) Eclipse est un environnement de développement intégré libre (le terme Eclipse désigne également le projet correspondant, lancé par IBM) extensible, et modulaire, qui permet potentiellement de créer des projets de développement pour le langage JAVA (et d'autres langages). Pour la particularité d'extensibilité ou (dis aussi portabilité), un programme écris avec le langage Java peut être compilé sur des plates-formes différentes: 64-bit Windows, Windows 10, Windows 7, Windows 8, Windows Vista, Windows XP, Linux, Solaris, … etc. L'autre caractéristique qui est la modularité permet d'écrire des portions de code génériques, c'est-à-dire utilisables par plusieurs applications. L'IDE Eclipse est principalement écrit en Java (à l'aide de la bibliothèque graphique : Standard Widget Toolkit (SWT), d'IBM), et ce langage, grâce à des bibliothèques spécifiques, est également utilisé pour écrire des extensions. La spécificité d' IDE Eclipse vient du fait que son architecture est totalement développée autour de la notion de plug-in.

L'un des principes phares de Java réside dans sa machine virtuelle : celle-ci assure à tous les développeurs Java qu'un programme sera utilisable avec tous les systèmes d'exploitation sur lesquels est installée une machine virtuelle Java. Lors de la phase de compilation de notre code source, celui-ci prend une forme intermédiaire appelée **byte code** : c'est le fameux code inintelligible pour votre machine, mais interprétable par la machine virtuelle Java. Cette dernière porte un nom : on parle plus communément de **JRE** (**J**ava **R**untime **E**nvironment). Plus besoin de se soucier des spécificités liées à tel ou tel OS (*Operating System*, soit système d'exploitation).

2. Installation de l'IDE Eclipse :

Vous devez installer le JDK (Java Development Kit). Pour ce faire, dans un moteur de recherche, vous tapez Télécharger JDK et vous accéder au site d'Oracle :

Java SE Development Kit 8 - Downloads - Oracle

Il existe deux façons de faire, soit télécharger Java Platform (JDK). Dans ce cas il faut faire une configuration qui est fastidieuse surtout pour un débutant et vous installez en plus un IDE (ex : Eclipse), c'est-à-dire un éditeur de codes pour développer en Java. L'autre méthode d'installation plus simple à comprendre par l'étudiant est de télécharger NetBeans with JDK. Si vous installez ce dernier, vous installez en même temps NetBeans, JDK et JRE (Java Runtime Environment), comme le montre la figure suivante :

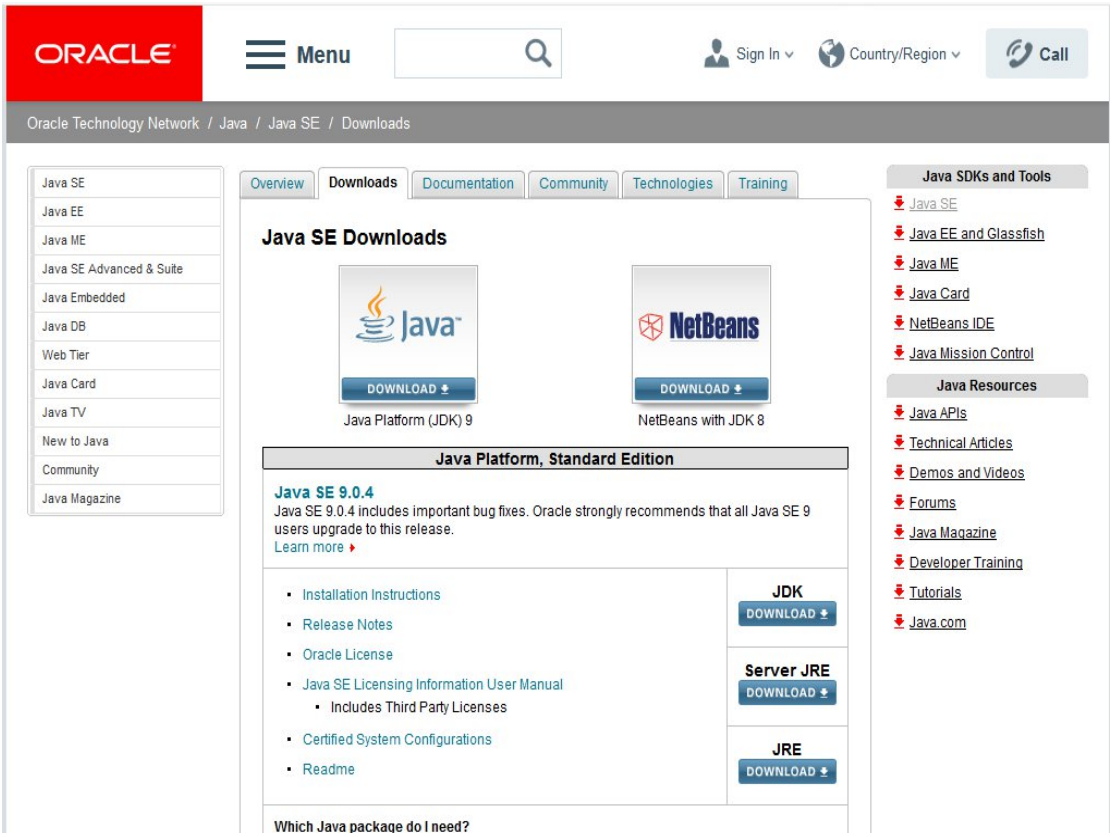


Figure 1. Installation Java. Java Platform (JDK) ou NetBeans.

Installation des deux logiciels nécessaires pour travailler avec Java :

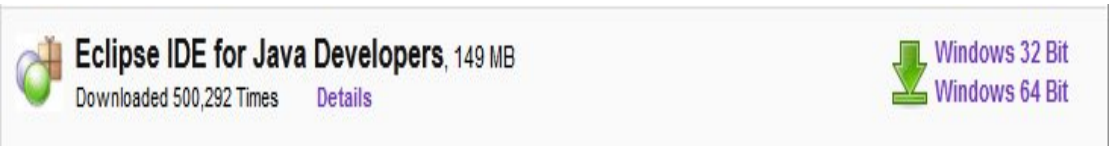
- L'utilitaire de developpement Java (Java SE Developpement Kit) [Cliquez sur Java SE Downloads]
- Editeur Java (Eclipse-Version 4.4.2 Luna)

Remarque :

-Il faut d'abord installer le JDK ensuite Ecplise
-Vous avez sans doute remarqué qu'on vous propose de télécharger soit le **JRE (J**ava **R**untime **E**nvironment), soit le **JDK** (**J**ava **D**evelopment **K**it). La différence entre ces deux environnements est que le JRE contient tout le nécessaire pour que vos programmes Java puissent être exécutés sur votre ordinateur; le JDK, en plus de contenir le JRE, contient tout le nécessaire pour développer et compiler.

Eclipse est un environnement de développement (IDE) historiquement destiné au langage Java. Ce logiciel permet de développer et de compiler des applications et des applets. Aussi, il permet de traduire les programmes en langage **byte code**.

Pour télécharger Eclipse IDE (https://www.eclipse.org), choisissez Eclipse IDE for Java Developers, en sélectionnant la version d'Eclipse correspondant à votre OS (*Operating System* = système d'exploitation), comme indiqué ci-dessous :



3. Utilisation de l'IDE Eclipse :

La partie suivante présente les outils de base de création des programmes avec Eclipse.

Lancement et « workspace » : Vous lancez le logiciel Eclipse. Un message apparait vous demandant de créer un espace de travail dans la fenêtre ''Workspace Launcher''. Le Workspace est un répertoire où Eclipse stocke les informations et les fichiers des projets Java crées. A l'ouverture, vous avez à gauche l'explorateur de fichiers et dossiers crées. Au milieu, l'espace d'édition de programmes. La partie en bas (Problems, Javadoc, Declaration), on ne s'en servira pas, donc vous pouvez fermer cette partie. A droite, la liste des taches peut être fermée pour avoir plus d'espace.

Création d'un projet Java : Dans JRE : vous choisissez **JavaSE-1.8**, et vous passez à l'étape suivante. On peut accéder au guide de création de projets par deux méthodes : par le menu « File/New/Java Project… » ou par le menu contextuel du le panneau « Package Explorer » (voir Figure 2).

Après avoir choisi « Project… », la fenêtre de création de projet (wizard) apparait (Figure 3a). Cette première étape permet de sélectionner quel type de projet l'on va créer. Pour créer votre premier projet Java, choisissez « Java Project » et puis « Next ». La seconde fenêtre (Figure 3b) permet de définir les propriétés essentielles du projet :

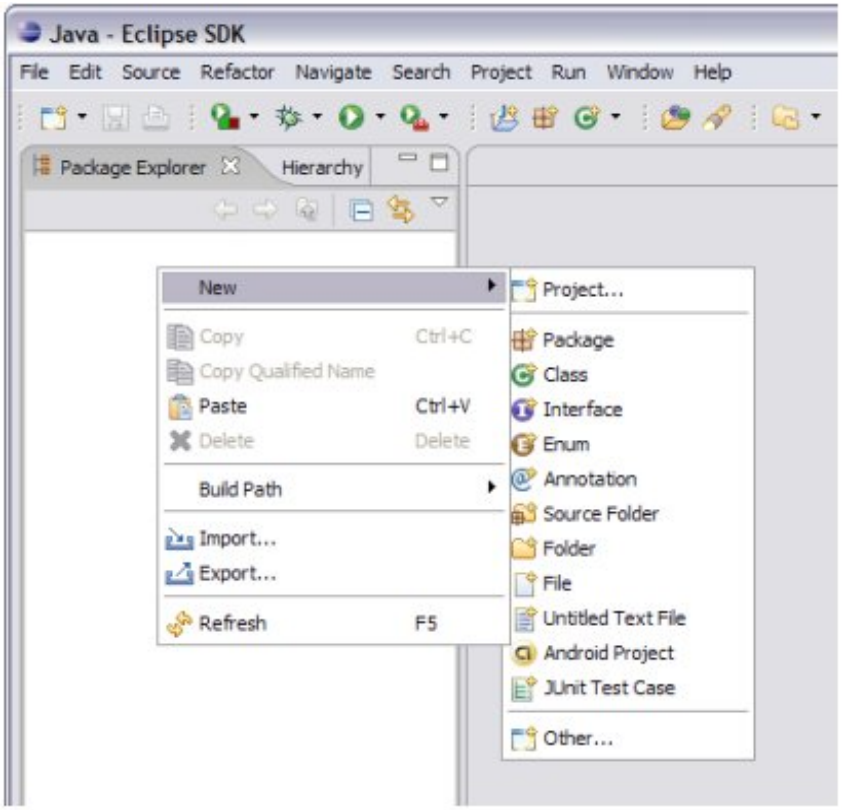


Figure 2. Création d'un projet. Menu contextuel du panneau "Package Explorer".

- ✓ Son nom (ici ProjetTest). Ce nom sera aussi utilisé par Eclipse comme nom de répertoire pour placer les fichiers du projet.
- ✓ L'endroit où l'on veut créer le projet (« Contents »). Il est possible de créer un nouveau projet dans le workspace en cours ou de créer un projet Eclipse à partir de code source Java déjà existant.
- ✓ La version de Java à utiliser pour compiler et exécuter le projet (« JRE »), ainsi que des options avancées de compatibilité entre les versions de Java.
- ✓ La structure du projet (« Project Layout »). Il y a deux manières de structurer les projets : placer tous les fichiers sources à la racine du projet ou séparer les sources et binaires dans des répertoires différents (« src » et « bin »). Ici, la deuxième option a été choisie car plus propre. A partir de ces premiers paramétrages, il est possible de cliquer sur « Finish » et de finaliser la création du projet.

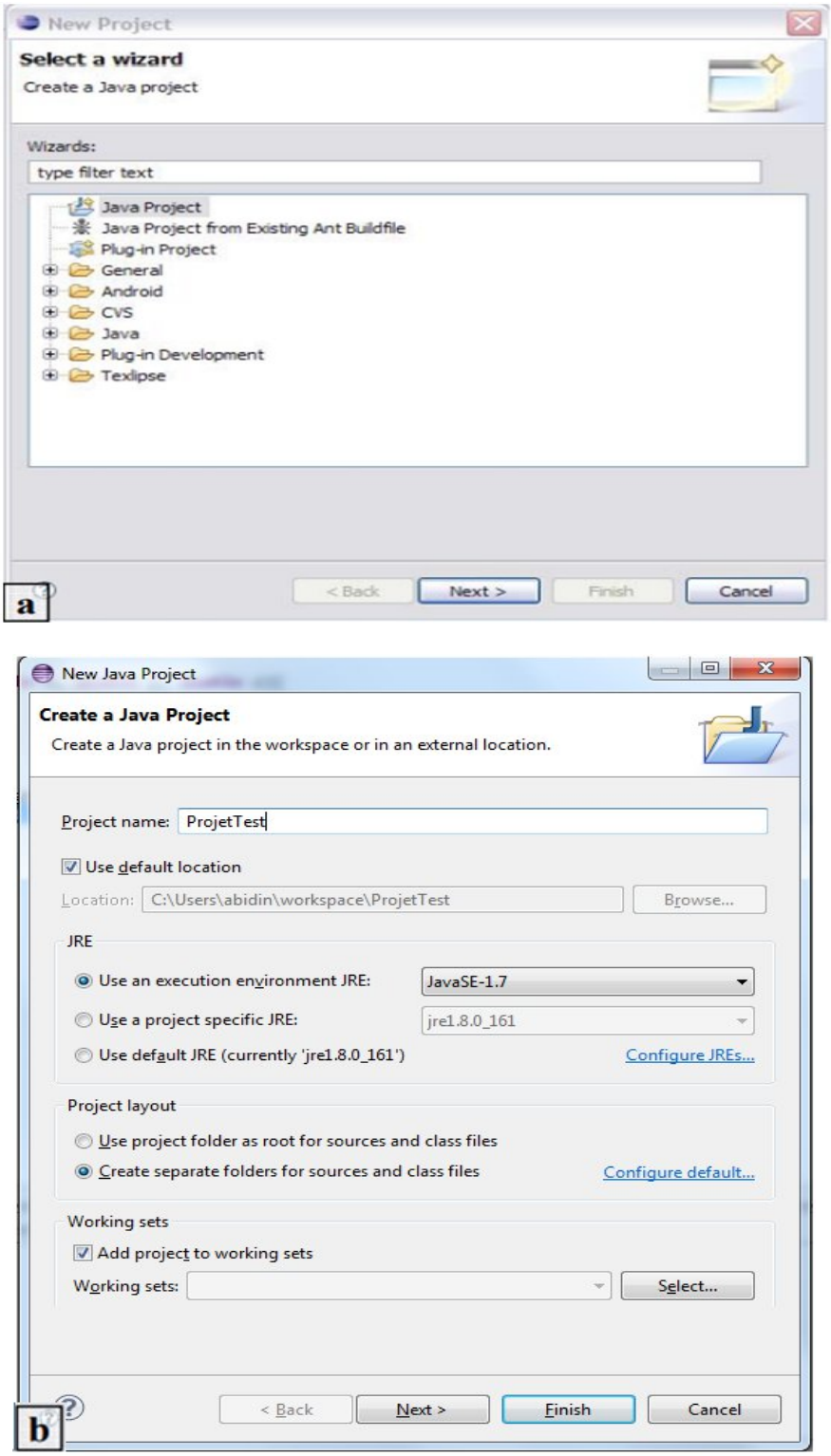


Figure 3. Création d'un projet - Wizard.

Création d'un package : Java permet de regrouper plusieurs classes dans un dossier appelé **package** afin de faciliter la modularité. Pour cela, il vous suffit d'inclure le dossier de votre package dans un projet et d'y importer les classes nécessaires.

Si l'instruction package était absente du fichier, alors c'est le package par défaut en Java ('**default package'**) qui est pris en considération. Ainsi toutes les classes du fichier vont appartenir au paquetage par défaut. L'instruction permettant de nommer un package doit figurer au début du fichier source (.java) comme suit :

----------Fichier Exemple.java----------

```
package nomtest ; // 1ere instruction
class MTest { }
class NTest { }
public class Exemple {// suite du code … }
```

----------Fin du Fichier ----------

Ainsi par cette opération nous venons d'assigner toutes les classes du fichier Exemple.java au package nomtest

Pour créer un nouveau package, cliquez simplement sur cette icône comme à la figure suivante **(vous pouvez aussi effectuer un clic droit puis : New > Package)** :

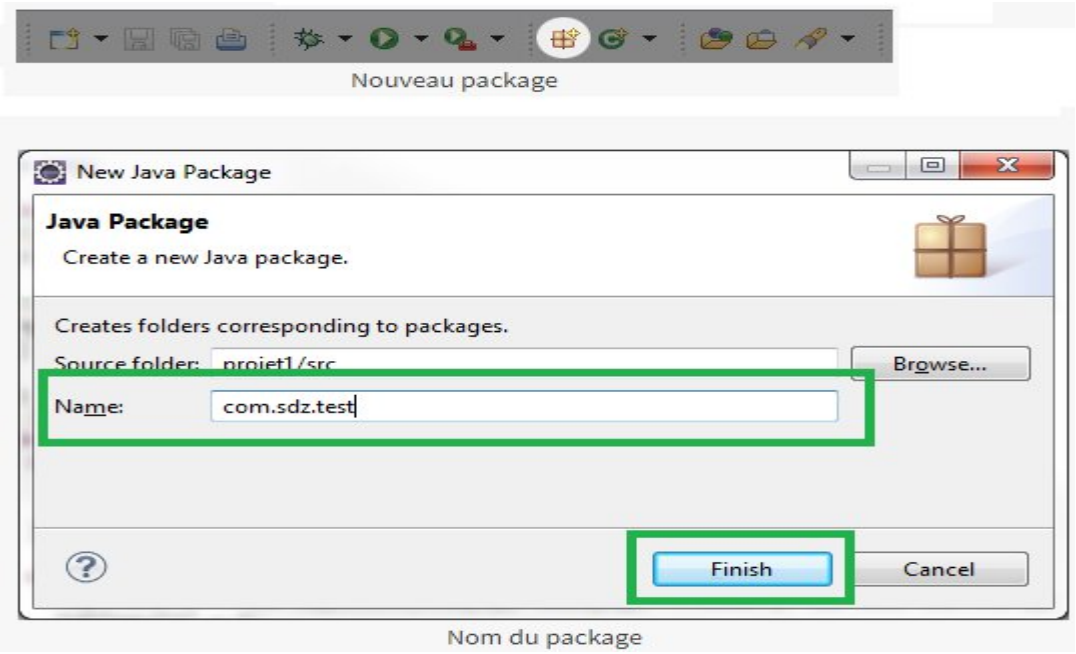


Figure 4. Création d'un Package.

Création d'une classe : Création de fichier avec le menu **« File/New/Class …»**, la fenêtre de création de classe apparait (Figure 4). Il faut spécifier le nom de la classe, et éventuellement : le package dans lequel elle se trouve. Dans le cas de création d'une classe principale '**main**'. En bas, vous devez cocher la case : **public static void** main(String[] args).

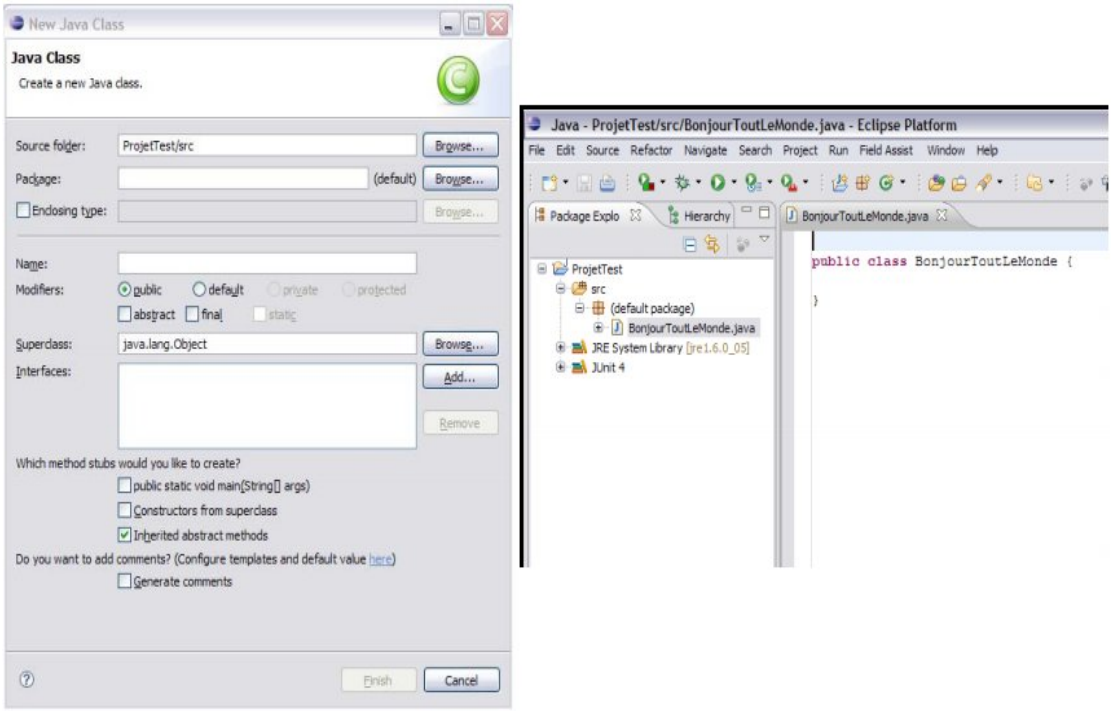


Figure 5. Création d'une nouvelle classe.

Dans cette section, l'étudiant va apprendre la création d'un nouveau projet nommé TP0 pour coder son premier programme en Java. Pour cela, choisir New ensuite Java Project. Vous retapez le programme ci-dessous en insérant l'instruction d'affichage d'un message : System.out.println (" Le message à afficher"), ensuite cliquez sur **Run**

```
/* Ce programme s'exécute dans la console d'Eclipse.
 * Il affiche le texte Bonjour tout le monde à l'ecran.
 */
public class Principale {  // Ne pas modifier cette ligne
      public static void main(String[] args) {
            // Cette méthode est le point d'entrée du programme
               System.out.println("Voici un programme simple JAVA");
      }
}
```

Quel est le rôle de la fonction System.*out*.println ?

4. Rappel des notions de base du langage Java :

- La syntaxe de java est très proche de celle du Langage C. On cite par exemple, qu'une affectation d'une valeur à une variable se fait avec l'operateur égal ('=').
- Toutes les instructions doivent se terminer par un point virgule (' ;')
- Les types de données de base en Java sont classifiées en plusieurs types : boolean, byte, short, int, long, float, double, char, string. L'espace mémoire requis pour chaque type de données est visualisé dans le tableau ci-dessous :

Type	Espace mémoire
boolean	indéterminé
char	2 octets (16 bits)
int	4 octets (32 bits)
byte	1 octet (8 bits)
short	2 octets (16 bits)
long	8 octets (64 bits)
float	4 octets (32 bits IEEE 754 floating point)
double	8 octets (64 bits IEEE 754 floating point))

- Le symbole **+** ici joue le rôle de concaténation et non pas la somme puis qu'il rassemble une variable **String** avec une variable de type **int**.
- Les **commentaires** sont des syntaxes qui servent à commenter son texte :
  - ✓ **//** commentaire sur une ligne
  - ✓ **/*** commentaire multi-lignes ***/**
- Pour déclarer une variable de type chaine de caractère on utilise la commande String suivie du nom de la variable
- Les noms de variables ne doivent contenir ni caractères accentués ni espaces et doivent, dans la mesure du possible, respectez la convention de nommage Java. Cette convention, la voici :
  - ✓ Tous les noms de classes doivent commencer par une majuscule ;
  - ✓ Tous les noms de variables doivent commencer par une minuscule ;
  - ✓ Si le nom d'une variable est composée de plusieurs mots, le premier commence par une minuscule, le ou les autres par une majuscule, et ce, sans séparation, par exemple : compteBancaire
- Lorsqu'on effectue des opérations sur des variables, on prend garde à leur type : On pourra perdre en précision.
- On peut *caster* un résultat en ajoutant un type devant celui-ci : (int), (double), etc.
- Les conditions vous permettent de n'exécuter que certains morceaux de code.
- Il existe plusieurs sortes de structures conditionnelles :

  La structure **if … elseif … else** ;

  La structure **switch … case … default** ;

  La structure **? :**
- Si un bloc d'instructions contient plus d'une ligne, vous devez l'entourer d'accolades afin de bien en délimiter le début et la fin.
- Pour pouvoir mettre une condition en place, on doit comparer des variables à l'aide d'operateurs logiques.
- On peut mettre autant de comparaisons renvoyant un **boolean** que vous le souhaitez dans une condition.
- Pour la structure **switch**, pensez à mettre les instructions **break** ; si on ne souhaite exécuter qu'un seul bloc **case**

- Pour créer des programmes interactifs acceptant les entrées d'un utilisateur, vous pouvez utiliser **System.in,** qui fait référence au périphérique d'entrée standard (généralement le clavier). Pour créer un objet Scanner et le connecter à l'objet **System.in,** vous écrivez une instruction similaire à celle-ci:

```
Scanner clavier = new Scanner(System.in);
```

  La classe **Scanner** contient des méthodes permettant d'extraire des valeurs d'un périphérique d'entrée. Cela signifie que les données sont acceptées lorsqu'un utilisateur appuie sur la touche Entrée, un espace ou une tabulation.
- La classe Scanner se trouve dans le package **java.util** que vous devrez importer. Pour faire ceci, vous devrez importer la classe **Scanner** grâce à l'instruction **import**. En introduisant la variable de type **Scanner** et en introduisant le point permettant d'appeler des méthodes de l'objet, Eclipse nous propose une liste de méthodes associées à cet objet (ceci s'appelle l'**autocomplétion**) comme le montre la figure ci-dessous ; de plus, lorsque on commence à taper le début de la méthode **nextInt()**, le choix se restreint jusqu'à ne laisser que cette seule méthode.

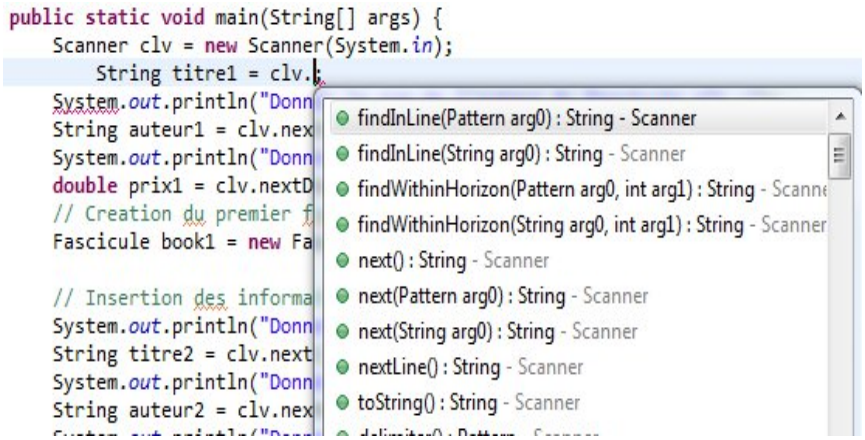


Figure 6. Exemple de l'Autocomplétion.

- Le tableau suivant récapitule certaines des méthodes les plus utiles pour lire différents types de données à partir du périphérique d'entrée par défaut :

Méthode	Description
nextDouble()	Récupère l'entrée en tant que Double
nextInt()	Récupère l'entrée en tant que int
nextLine()	Récupère la ligne de données suivante et la renvoie sous forme de chaîne (String)
next()	Récupère la prochaine entrée sous forme de chaîne
nextShort()	Récupère l'entrée en tant que Short
nextByte()	Récupère l'entrée en tant que octet (Byte)
nextFloat()	Récupère les entrées sous forme de float. Notez que lorsque vous entrez une valeur d'entrée qui sera stockée sous forme de float, vous ne tapez pas de F. Le F est utilisé uniquement avec les constantes codées dans un programme.

Méthode	Description
nextLong()	Récupère l'entrée en tant que long. Notez que lorsque vous entrez une valeur d'entrée qui sera stockée longtemps, vous ne tapez pas un L. Le L est utilisé uniquement avec des constantes codées dans un programme.

- Il y a une méthode de récupération de données pour chaque type **(sauf les char) :** nextLine() pour les **String**, nextInt() pour les **int**. La classe Scanner ne contient pas de méthode **nextChar()**. Pour extraire un seul caractère du clavier, il faut utiliser la méthode **nextLine()**, puis la méthode **charAt()**.
- En java, comme en tout langage de programmation, il est nécessaire, d'afficher de temps en temps une ou plusieurs informations à l'écran. En java, on utilise pour cela la fonction pré-définie **System.out.print()**. **print()** est une fonction de sortie standard utilisée dans Java qui appartient à la classe **System.out**. La classe qui commence toujours par une lettre majuscule, **out** spécifie le nom de la classe et **print()**. La méthode **print()** qui appartient aussi à la classe **System.out**, est similaire à la méthode **println()**, cependant elle ne fait pas de retour à la ligne. Les méthodes sont invoquées au moyen de **paramètres**, ces derniers étant des messages envoyés à la méthode.

Exercice 1 :

Ecrire un programme principal qui calcule le quotient de deux entiers donnés et l'affiche dans la console. Ce programme sera divisé en trois étapes :

1- Déclaration des variables (dividende, diviseur, quotient) et initialisation des variables (dividende=5, diviseur=2)
2- Calcul du quotient
3- Affichage du résultat dans la console

- Est-ce que votre programme donne le bon résultat ? Expliquez pourquoi ?
- Corrigez votre programme afin qu'il fournisse le bon résultat

Exercice 2 : (Instructions conditionnelles)

Note :
- Instruction : Si… Alors … Sinon…
If (*condition*) {
actions
} else {
actions
}
-Operateurs de comparaison: égalité ==, différence !=

Ecrire un programme principal Java qui permet de saisir des nombres entiers à partir du clavier. Ensuite faire un test pour afficher si le nombre X est négatif, nul, ou positif.

Note : Pour bien structurer votre programme final, allez dans : **Source-- Format**

Exercice 3 :

- Ecrire un programme principal Java qui calcule le prix d'un article (prixArticle) soldé selon qu'il soit soldé à -20% ou -50%. Donc on demande à l'utilisateur de faire la saisie du prix de l'article avant qu'il soit soldé.
- Ensuite de rentrer le code solde selon que l'article est soldé -20% (codeSolde=1) ou -50% (codeSolde=2) et calculer le prix de l'article soldé, puis l'afficher.
- Dans le même programme, vous rajoutez un cas supplémentaire qui consiste à ce que l'article soit soldé à -70% sans taper le codeSolde correspondant dans le programme.
- Remplacez fonction System.*out*.print par la fonction System.*out*.println, quelle est votre remarque ?

Exercice 4 : (Branchement conditionnel)

Dans le cas de choix multiples, il n'est pas forcément très pratique d'enchainer les 'if …else…'. Il est donc plus judicieux d'utiliser l'instruction à choix multiples **Switch.** Un **case** représente un point à partir duquel l'exécution du code commencera. Si on veut isoler chaque bloc de cas, il faut utiliser une instruction **break**. Au contraire, l'omission de l'instruction **break** peut être pratique si on veut effectuer le même traitement pour un ensemble de cas.

switch (*nomVariable*) **{** // *nomVariable* est une variable de type **int** ou **char**

case *valeur1* : *actions*
break ;
case *valeur2* : *actions*
break ;
default : *actions* // exécuté si *nomVariable* ne prend aucune des valeurs proposées
} // CAS FACULTATIF

- A refaire les exercices 5 et 6 en utilisant cette fonction à choix multiples.

Exercice 5 : (Instructions itératives =Boucles)

For **(***nomVariable* = valeur Debut **;** <=valeurFin ; nomVariable**++){**
actions // *nomVariable* est une variable de type **int**
}
Ou bien (par exemple)
For (*nomVariable = valeurDebut* ; *nomVariable* > *valeurFin* **;** *nomVariable***--){**
actions
}

Avec **: nomVariable++** incrémente la variable courante de 1, alors l'expression vaut la valeur de **nomVariable** avant incrémentation.
Et nomVariable-- décrémente la variable courante de 1, alors l'expression vaut la valeur de **nomVariable** avant décrémentation.

Dans un cas général où l'opérateur (**++** ou **--**) précède la variable, la valeur de l'expression est la valeur de la variable après incrémentation ou décrémentation.

- Ecrire un programme Java qui permet de saisir un nombre entier à partir du clavier.
- Ecrire le code qui calcule le factoriel de ce nombre entier *n*
- Afficher le résultat ('Factoriel de *n* est …')

TP1 : Classes et Objets Java avec Eclipse

1. Introduction :

Une classe est le support de l'encapsulation : c'est un ensemble de données et de fonctions regroupées dans une même entité. Une classe est une description abstraite d'un objet. Les fonctions qui opèrent sur les données sont appelées des méthodes. Instancier une classe consiste à créer un objet sur son modèle. Entre classe et objet il y a, en quelque sorte, le même rapport qu'entre type et variable.
Java est un langage orienté objet : tout appartient à une classe sauf les variables de types primitives. Pour accéder à une classe il faut en déclarer une instance de classe ou objet. Une classe comporte sa déclaration, des variables et les définitions de ses méthodes.
Une classe se compose de deux parties : un en-tête et un corps. Le corps peut être divisé en deux sections : la déclaration des données et des constantes et la définition des méthodes. Les méthodes et les données sont pourvues d'attributs de visibilité qui gèrent leur accessibilité par les composants hors de la classe.

2. Types de variables et de classes :

Le principal avantage avec la programmation en langage Java vient d'une part du fait que l'on peut :

- Créer des objets
- Libérer des objets non utilisés à l'aide d'un ramasse-miettes

Un objet est une instance d'une classe, qui est l'outil fondamental de programmation en Java. Pour pouvoir manipuler des objets, il est essentiel de définir des classes, c'est-à-dire définir la structure d'un objet. Les classes sont composées de méthodes et d'attributs qui peuvent être *public*, *private* ou *protected*.

Les différents mots clés concernant l'accessibilité sont les suivants :

- **Aucun mot clé**: accessible par les classes du même paquetage ou *package* ;
- **public**: accessible par toutes les classes ;
- **protected**: accessibles par toutes les classes héritées et les classes du même paquetage, inaccessibles par les autres ;
- **private**: inaccessible par toute autre classe.

Il y a deux types de variables (attributs) en programmation Java :

- **Variables d'instance** : valeurs propres à l'objet, elles doivent être non accessibles par les objets des autres classes.
- **Variables de classe** : valeurs communes à tous les objets d'une même classe (variables globales aux objets de la classe) : valeurs non réinitialisées à chaque nouvel objet, elles sont statiques (static). Elles peuvent être publiques (public) ou privées (private).

3. Rappel :

- Une **classe** (classe = catégorie d'objets) correspond aux concepts d'abstraction et d'encapsulation
- C'est une abstraction car elle présente une interface (ensemble de méthodes) qui permet d'interagir avec les objets de cette classe.
- Une classe encapsule des données (ensemble d'attributs)
- Une classe définit un **type**
- Une réalisation particulière d'une classe s'appelle une **instance**
- instance= **objet**
- Un objet est une **variable** de type de sa classe

En Java une **classe** se déclare par le mot-clé **class**

Exemple **:** **class Rectangle** { … }

Ceci définit un nouveau **type** du langage.
La déclaration d'une **instance** d'une clase se fait de façon similaire à la déclaration d'une **variable :** ***nomClasse nomInstance* ;**

Exemple : **Rectangle rect1 ;**

Cet exemple déclare une instance **rect1** de la classe **Rectangle**.

Rappelez-vous la première partie dans le TP0 : vous avez déjà utilisé des objets sans vous en rendre compte, lorsque vous faisiez ceci : **Scanner** clavier = new **Scanner**(System.*in*).

Ici **clavier** est un objet de type **Scanner**. Vous avez utilisé un objet de la classe **Scanner**: on dit que vous avez créé une *instance* de la classe **Scanner()**.

Le mot-clé '**this'** désigne, dans une classe, l'instance courante de la classe elle-même. **'this'** est absolument nécessaire lorsque le nom de l'attribut et celui du paramètre sont les mêmes. On peut désigner explicitement l'attribut dans une classe grâce à **'this'** :

```
public class Calculateur
{
   protected int valeur ;
   public void calcule (int valeur){
    this.valeur = this.valeur + valeur;
                                  }
}
```

Dans cet exemple, la méthode **calcule** additionne le paramètre **valeur** à l'attribut **valeur** et stocke le résultat dans l'attribut **valeur**. L'attribut a été désigné explicitement par le mot-clé **this**, désignant l'instance de la classe, préfixé au nom de l'attribut.

4. Objectifs :

- Définir les propriétés et méthodes d'une classe
- Créer une instance de classe
- Appliquer des méthodes
- Utiliser les variables de classe (static)

Exercice 1 :

- Exécuter le programme ci-dessous
- Qu'affiche l'exécution de ce programme ?

```
public class Principale {  // Ne pas modifier cette ligne
      public static void main(String[] args) {
            // Déclaration des variables
            int diviseur;
            double dividende, quotient;
            // Enregistrement des valeurs tapées par l'utilisateur
            // Création d'un objet de la classe Scanner nommé clavier
            Scanner clavier= new Scanner(System.in);
            System.out.print("Entrer le dividende : ");
            // Enregistrement d'un nombre décimal tapé au clavier
            dividende=clavier.nextDouble();
            System.out.print("Entrer le diviseur : ");
            // Enregistrement d'un nombre entier tapé au clavier
            diviseur=clavier.nextInt();
            // Calcul du quotient
            quotient= dividende / diviseur;
            // Affichage du résultat dans la console
            System.out.print("Le quotient de la division est " +
quotient);
            clavier.close();  // Libération de l'espace mémoire utilisé
par le scanner (ou réservé a l'objet clavier)
      }
}
```

Exercice 2 :

1- Définir une classe **Avion** avec les attributs suivants : *Id, Marque, Vitesse, Puissance*.
2- Définir un **constructeur** dans la classe **Avion** permettant d'initialiser les attributs d'un objet avion. Sachant que : Id doit être auto-incrément.
3- Afficher les informations de la classe **Avion** en utilisant une méthode de type **void**, qu'on appelle **afficher()**.
4- Donnez la définition des variables statiques.
5- Créer une classe main pour tester la classe **Avion** qui consiste à afficher l'exemple d'exécution ci-dessous.

Exemple d'exécution :

id=1, marque= Boeing, vitesse=920.0 km/h, puissance=_3000
id=2, marque=ATR, vitesse=554.0 km/h, puissance=2750
id=3, marque=Airbus, vitesse=1020.0 km/h, puissance=3700
Le nombre d'avions est : 3

Exercice 3 :

1- Créer une classe Java de visibilité public nommée **Tirelire** qui représente un récipient où on introduit les pièces de monnaie, ayant pour attribut : **public Double montant**
2- Créer un constructeur ayant comme paramètre montant.
3- Créer une méthode **deposer()** du type **void** qui gère les versements.
4- Créer une méthode **retirer()** du type **void** qui gère les retraits.
5- Créer une méthode **afficher()** du type **void** permettant d'afficher le montant
6- Créer une classe **TestTirelire** permettant d'afficher le montant de la tirelire sachant qu'on effectue un versement de 2000 Da, et puis un retrait de 1000 Da avec un montant initial de 5000 Da.

Exercice 4 :

On donne la classe principale main **TestQuadrilatère** pour afficher les résultats surface et périmètre avec une Longueur=7 et une Largeur=5 comme suit :

```
// Création d'une classe TestQuadrilatère pour afficher les résultats
surface et périmètre

public class TestQuadrilatère {
        // Programme principal
        public static void main(String[] args) {
        // Création de l'objet monQuadrilatère
                Quadrilatère monQuadrilatère = new Quadrilatère(7.0, 5.0);
// Affichage des résultats de la surface et du périmètre
                monQuadrilatère.afficher();
        }
}
```

1- Créer la classe **Quadrilatère** ainsi que les méthodes correspondantes afin de permettre le fonctionnement du programme principal de test **TestQuadrilatère**.
2- Définir l'étape d'**instanciation** et à quel niveau du programme se fait cette étape ?

Exercice 5 :

1- Définir une classe **Fascicule** avec les attributs suivants : *Id, Titre, Auteur (Nom complet), Prix*.
2- Définir un **constructeur** permettant d'initialiser les attributs d'un objet fascicule par des valeurs saisies par l'utilisateur. Sachant que Id doit être auto-incrément.
3- Définir la méthode **toString()** permettant d'afficher les informations du fascicule en cours.
4- Créer une classe main pour tester la classe **Fascicule** qui consiste à afficher l'exemple d'exécution ci-dessous.

Exemple d'exécution :

Donner le titre du fascicule n° 1 : C++
Donner le nom de l'auteur du fascicule n° 1 : MOHAMED
Donner le prix du fascicule n° 1 :200
Donner le titre du fascicule n° 2 : JAVA
Donner le nom de l'auteur du fascicule n° 2 : ABIDINE

Donner le prix du fascicule n° 2 : 300
Donner le titre du fascicule n° 3 : Python
Donner le nom de l'auteur du fascicule n° 3 : FERGANI
Donner le prix du fascicule n° 3 : 250
Le prix du fascicule Langage C++ de l'auteur nommé MOHAMED est :200 Da
Le prix du fascicule Langage JAVA de l'auteur nommé ABIDINE est :300 Da
Le prix du fascicule Langage Python de l'auteur nommé FERGANI est :250 Da
Le nombre de fascicules est :3

TP2 : Méthodes d'accès aux attributs: Getters et Setters

1. Introduction :

Dans ce TP, la notion des accesseurs (getters) et des mutateurs (setters) en utilisant plusieurs classes sera abordée. Un accesseur est une méthode qui va nous permettre de récupérer le contenu d'une donnée membre protégée en lecture. Un **accesseur**, pour accomplir sa fonction :

- doit avoir comme type de retour le type de la variable à renvoyer
- ne doit pas nécessairement posséder d'arguments

Un **mutateur** est une méthode permettant de modifier le contenu d'une donnée membre protégée en écriture. Un mutateur, pour accomplir sa fonction :

- doit avoir comme paramètre la valeur à assigner à la donnée membre. Le paramètre doit donc être du type de la donnée membre
- ne doit pas nécessairement renvoyer de valeur (il possède dans sa plus simple expression le type *void*)

Grâce aux accesseurs, vous pouvez lire les variables de vos objets, et grâce aux mutateurs, vous pouvez les modifier.

En principe, toutes les variables d'un objet sont privées : seul l'objet auquel elles appartiennent peut y accéder. Pour les atteindre depuis l'extérieur, il faut utiliser les accesseurs et les mutateurs. Pour pouvoir être appelés depuis l'extérieur, accesseurs et mutateurs doivent être des méthodes publiques.

2. Rappel :

Il y a une convention de nommage standard des méthodes pour représenter les accès à des attributs, afin de faire ressortir leurs fonctions premières*:*

- Pour l'accesseur le nom de la méthode commence par le préfixe **get**. Il retourne la valeur d'une variable d'instance ou de classe. On note qu'une variable de classe est toujours déclarée « **static** », est indépendante des objets instanciés et pourrait servir à partager des informations entre les instances de cette classe.
- Pour le mutateur, le nom de la méthode commence par le préfixe **set**. Elle peut modifier directement la valeur d'une variable d'instances ou de classe avec un type du retour de type **void**.
- Nos méthodes **accesseurs** et **mutateurs** doivent bien évidemment être **publiques** afin d'être accessibles depuis les autres classes (sans quoi il n'y aurait pas d'intérêt).
- Pour créer les méthodes **accesseurs** et **mutateurs**, choisir le menu **« Source/Generate Getters and Setters …»**, comme le montre la figure ci-dessous :

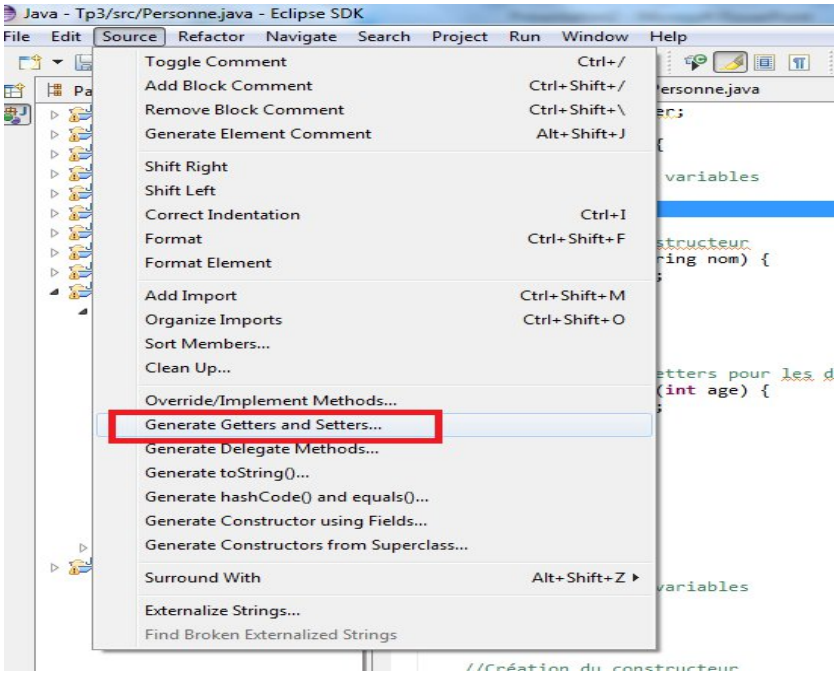


Figure 1. Création des **accesseurs** et **mutateurs**

Exemple d'application:

```
public class Ville {

  //Les variables et les constructeurs n'ont pas changé
  //*************   ACCESSEURS *************
  //Retourne le nom de la ville
  public String getNom()  {
    return nomVille;
  }
  //Retourne le nom du pays
  public String getNomPays()
  {
    return nomPays;
  }
  // Retourne le nombre d'habitants
  public int getNombreHabitants()
  {
    return nbreHabitants;
  }

  //*************   MUTATEURS   *************
  //Définit le nom de la ville
  public void setNom(String pNom)
  {
    nomVille = pNom;
  }
  //Définit le nom du pays
  public void setNomPays(String pPays)
  {
    nomPays = pPays;
  }
  //Définit le nombre d'habitants
  public void setNombreHabitants(int nbre)
  {
    nbreHabitants = nbre;
  }
}
```

3. Objectifs :

- Définir les propriétés et méthodes d'une classe
- Définir des constructeurs
- Créer une instance de classe
- Accéder par les accesseurs et mutateurs aux propriétés en lecture et écriture d'un objet
- Appliquer des méthodes

Exercice 1 :

- Ecrire un programme principal qui permet d'entrer le Nom, Prénom, Job, Age, pour l'inscription d'une personne et de pouvoir les changer à chaque fois en utilisant les mutateurs.

  Exemple d'exécution :
  Début de l'inscription
  Votre nom : Rayan
  Votre prénom : Diab
  Votre Job : Consultant
  Votre âge : 30
- Créer une méthode **afficher()** du type **void** pour afficher les résultats
- Changer maintenant l'âge de la personne avec les mutateurs et afficher le message suivant ('Accès interdit aux mineurs !') si l'âge inséré est inferieur à 18 ans.

Exercice 2 :

- Créer une classe Java nommée **Quadrilatère** ayant pour attributs de type privé : Longueur et Largeur tous du type Double, ajouter ensuite un constructeur avec paramètres.
- Accéder par les mutateurs aux propriétés en écriture d'un objet
- Définir une méthode **surface()** du type Double permettant de calculer la surface du quadrilatère
- Définir une méthode **périmètre()** du type Double permettant de calculer le périmètre du quadrilatère
- Créer une méthode **afficher()** du type **void** pour afficher les résultats. Donner ensuite le code complet de la classe.
- Créer une classe main **TestQuadrilatère** pour afficher les résultats surface et périmètre.

Exercice 3 :

- Définir une classe **Fascicule** avec les attributs suivants de type privé : *Id, Titre, Auteur (Nom complet), Prix*.
- Accéder par les accesseurs aux propriétés en lecture d'un objet fascicule

- Définir un **constructeur** permettant d'initialiser les attributs d'un objet fascicule par des valeurs saisies par l'utilisateur. Sachant que Id doit être auto-incrément.
- Définir la méthode **toString ()** permettant d'afficher les informations du fascicule en cours.
- Écrire un programme testant la classe **Fascicule** et son exécution permet d'afficher sur la console l'exemple suivant :

Exemple d'exécution :
Combien de livre voulez vous créer ?
Donner le titre du fascicule : C++
Donner le nom de l'auteur du fascicule : MOHAMED
Donner le prix du fascicule :200
Donner le titre du fascicule : JAVA
Donner le nom de l'auteur du fascicule : ABIDINE
Donner le prix du fascicule : 300
Donner le titre du fascicule : Python
Donner le nom de l'auteur du fascicule : FERGANI
Donner le prix du fascicule : 250
L'auteur du fascicule n° 1 s'appelle : MOHAMED
Le prix de ce fascicule est de: 200Da
Ce fascicule a un titre : C++
L'auteur du fascicule n° 2 s'appelle : ABIDINE
Le prix de ce fascicule est de: 300Da
Ce fascicule a un titre : JAVA
L'auteur du fascicule n° 3 s'appelle : FERGANI
Le prix de ce fascicule est de: 250Da
Ce fascicule a un titre : Python

Exercice 4 :

- On définit une classe **Point** où chaque point est défini par les coordonnées (x, y) de type **int**.
- Calculer la distance euclidienne entre deux points de coordonnées A(10, 5) et B(12, 7) en créant un constructeur pour les coordonnées des deux points et deux méthodes **distance()** et **affichage()**

La méthode **distance()** calcule la distance Euclidienne entre les deux points A et B. La méthode **affichage()** permet l'affichage de cette distance.

- Si cette distance correspond à un rayon d'un cercle
    - Calculer le périmètre du cercle avec **getPerimetre()**.
    - Calculer la surface du cercle avec **getSurface()**.
- Changer la valeur du premier point A(2,2) avec les mutateurs et recalculer la distance euclidienne

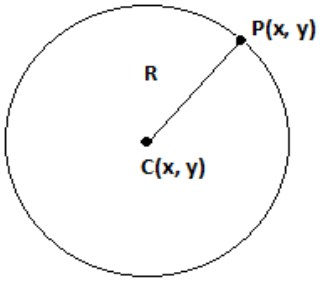
Exercice 5 :
Un cercle est défini par un point de coordonnées (x, y) qui représente son centre : **centre(x, y)** et un **rayon**. On définit le type d'objet crée par la classe, dans notre cas le type est un **Point** caractérisé par ces coordonnées (à deux dimensions (x, y)).

Créer la classe **Point** définie par :

- Les attributs **x** et **y** de type **int**
- Un constructeur qui initialise les valeurs de **x** et **y**
- Une méthode **toString()** : retourne une chaine de caractères de type **Point(x,y)**

On peut créer un cercle de deux manières comme le montre la figure ci-dessous :

- Soit en précisant son centre et un point du cercle
- Soit en précisant son centre et son rayon



Vous devriez implémenter les méthodes à exécuter sur un cercle dans la classe **Cercle** qui sont :

- **getPerimetre()** : retourne le périmètre du cercle
- **getSurface()** : retourne le périmètre du cercle
- **appartient(Point p)** : retourne si le point **p** appartient ou non à l'intérieur du cercle
- **toString()** : retourne une chaine de caractères de type **CERCLE(x,y,R)**

Créer une application **test** qui permet de :

- Créer un cercle défini par le centre c(100, 100) et un point p(200, 200)
- Créer un cercle défini par le centre c(130, 100) et de rayon r=40
- Afficher le périmètre et le rayon des deux cercles
- Afficher si le point p(120, 100) appartient à l'intersection des deux cercles ou non.

TP3 : Tableaux et vecteurs

1. Introduction :

Dans ce TP, la notion des tableaux en utilisant plusieurs classes sera abordée. Vous allez en effet pouvoir lui affecter plusieurs valeurs ordonnées séquentiellement que vous pouvez appeler au moyen d'un indice (ou d'un compteur, en utilisant les boucles). Il vous suffira d'introduire l'emplacement du contenu désiré dans notre variable tableau pour la lecture et l'affichage.

2. Rappel :

2.1 Tableaux statiques :

Un tableau est un **objet.** Ce type d'objet est contenu dans le package par défaut (pas d'import nécessaire). Il est appelé statique parce qu'il a une longueur fixe.
La déclaration ressemble beaucoup à celle d'une variable quelconque, si ce n'est la présence de crochets « [ ] » après le nom de notre tableau et d'accolades « { } » encadrant l'initialisation de celui-ci.

typeElements nomTableau []= new *typeElements* [Nombre_Elements] ;

Dans la pratique, ça nous donnerait quelque chose comme ceci :

int tableauEntier[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
double tableauDouble[] = {0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0};
char tableauCaractere[] = {'a', 'b', 'c', 'd', 'e', 'f', 'g'};
String tableauChaine[] = {"chaine1", "chaine2", "chaine3" , "chaine4"};

Vous remarquez bien que la déclaration et l'initialisation d'un tableau se font comme avec une variable ordinaire : il faut utiliser des « ' ' » pour initialiser un tableau de caractères, des « " " » pour initialiser un tableau de String, etc. Vous pouvez aussi déclarer un tableau vide, mais celui-ci devra impérativement contenir un nombre de cases bien défini. Par exemple, si vous voulez un tableau vide de six entiers :
int tableauEntier[] = new int[6];
//Ou encore
int[] tableauEntier2 = new int[6];

Lecture de valeurs d'un tableau :
nomVariable = *nomTableau* [*position*]; // La 1ère case du tableau a l'indice 0

Nombre de cases d'un tableau :
nomTableau.**length** // nombre de cases du tableau

En Java, les tableaux à plusieurs dimensions sont en fait des tableaux de tableaux.
Un tableau de tableaux d'entiers a pour type **« int [] [] »**
On peut créer tous les tableaux sans les initialiser :
int [] [] tableau = new int [n] [m] ;
Cette instruction crée un tableau de **n** lignes, chaque case contenant un tableau de **m** colonnes.
tableau[i] [j] donne l'accès à l'élément **j** de la ligne **i**.

2.2 Tableaux dynamiques :

Un tableau dynamique est un tableau dont la longueur n'est pas fixée.
C'est un objet de la classe **ArrayList** qui ne peut contenir que des objets de type complexe.
Construction d'un tableau dynamique vide :
ArrayList <typeElements> nomTableau = new ArrayList <typeElements>() ;

Il existe une autre classe java '**Vector**' qui implémente un tableau d'objets similaire à **ArrayList**. **Vector** implémente l'interface List comme **ArrayList**, son inconvénient est qu'il donne de faible performance à cause de sa synchronisation dans les opérations d'ajout, recherche, suppression et modification de ces éléments. Il est recommandé d'utiliser **ArrayList**.

Les principales méthodes de la classe ArrayList sont :
nomTableau.size() // renvoie la taille du tableau
nomTableau.get(i) // renvoie l'élément d'indice **i** du tableau (1er élément en position 0)
nomTableau.set(i, valeur) // remplace la valeur de la case i du tableau par valeur
nomTableau.remove(i) // supprime l'élément d'indice **i** du tableau et déclare tous les éléments du tableau après l'indice **i** d'un rang vers la gauche
nomTableau.add (valeur) // ajoute des éléments à un tableau

On cite deux méthodes couramment utilisées dans la programmation dynamique qui sont : **toString()** et **equals()** pour les objets.

La méthode **toString()** permet de décrire les propriétés d'objets d'une classe sous la forme d'une chaîne de caractères.

La méthode **equals** pour les objets est identique à la comparaison (**==**) dans sa version par défaut. Elle sert à comparer les champs de deux objets.

Exercice 1 :

- Définir une classe **Avion** avec les attributs suivants : *Id,* Marque*, Vitesse, Puissance*.
- Définir un **constructeur** permettant d'initialiser les attributs d'un objet avion par des valeurs passées en paramètre. Sachant que Id doit être auto-incrément.
- Définir une méthode **Afficher ()** permettant d'afficher les informations d'un avion.
- Ecrire un programme testant la classe main **TestAvion** en utilisant un tableau nommé **avions** de taille=3 et son exécution permet d'afficher sur la console l'exemple suivant :

Exemple d'exécution :
id=1, marque= Boeing, vitesse=920.0 km/h, puissance=_3000
id=2, marque=ATR, vitesse=554.0 km/h, puissance=2750
id=3, marque=Airbus, vitesse=1020.0 km/h, puissance=3700
Le nombre d'avions est : 3

Exercice 2 :

- Définir une classe **Fascicule** avec les attributs suivants : *Id, Titre, Auteur (Nom complet), Prix*.
- Définir un **constructeur** permettant d'initialiser les attributs d'un objet fascicule par des valeurs saisies par l'utilisateur. Sachant que Id doit être auto-incrément.
- Définir la méthode **Afficher ()** permettant d'afficher les informations du fascicule en cours.
- Ecrire un programme testant la classe main **TestFascicules** en utilisant un tableau nommé **fascicules** de taille=3 et son exécution permet d'afficher sur la console l'exemple suivant :

Exemple d'exécution :
Donner le titre du fascicule n° 1 : C++
Donner le nom de l'auteur du fascicule n° 1 : MOHAMED
Donner le prix du fascicule n° 1 :200
Donner le titre du fascicule n° 2 : JAVA
Donner le nom de l'auteur du fascicule n° 2 : ABIDINE
Donner le prix du fascicule n° 2 : 300
Donner le titre du fascicule n° 3 : Python
Donner le nom de l'auteur du fascicule n° 3 : FERGANI
Donner le prix du fascicule n° 3 : 250
Le prix du fascicule Langage C++ de l'auteur nommé MOHAMED est :200 Da
Le prix du fascicule Langage JAVA de l'auteur nommé ABIDINE est :300 Da
Le prix du fascicule Langage Python de l'auteur nommé FERGANI est :250 Da
Le nombre de fascicules est :3

Exercice 3 :
Un capital est placé durant *n* années à un taux annuel d'intérêts composés de *t* %.
Réaliser un seul programme main qui calcule le capital et les intérêts acquis au bout des *n* années. On utilise deux tableaux pour enregistrer le capital et les intérêts acquis chaque année puis les afficher.

Exercice 4 :
Créer un programme main qui permet la saisie de plusieurs températures entières, forcement entre -20 et +40 (il faut contrôler la saisie). Le nombre de températures doit être saisi dès le début du programme. Au final, pour chaque température, il faut afficher sa fréquence (le nombre de fois où cette température a été saisie).

Exercice 5 :
Créez une classe nommée **Corpulence** d'un individu qui possède :
- Trois champs **nom**, **taille** (en cm) et **poids** (en Kg)
- Un constructeur par défaut et un constructeur d'initialisation avec les paramètres **nom**, **taille** et **poids**
- Deux méthodes **calculImc** et **analyseImc** de l'objet courant
  Le calcul de l'imc d'un individu est comme suit : imc= poids[kg] /taille^2[m]
  Pour analyser l'imc, on suppose que pour chaque cas, on aura le renvoi de chaine de caractères comme suit :

imc < 16 "Anorexie"
16 ≤ imc < 20 "Maigreur"
20 ≤ imc < 26 "Normal"
26 ≤ imc < 30 "Surpoids"
30 ≤ imc "Obésité"

- Définir la méthode **toString ()** permettant de renvoyer une chaine de caractères nom, taille, poids, et la valeur donnée par la méthode **analyseImc**.
- Créer une méthode **equals()** qui sert à comparer si deux objets de classe **Corpulence** ont les mêmes champs (nom, taille, poids)
- Créer la classe **Test** en utilisant un tableau dynamique contenant les instances de la classe **Corpulence** dans lequel on place 3 objets que l'on crée arbitrairement, puis on affiche le contenu du tableau à l'aide de la méthode **toString()**
- Créer un quatrième objet avec les mêmes champs du deuxième objet crée et déterminer qu'ils sont bien similaires afin de supprimer l'enregistrement qui se répète.

Exercice 6 :

- Créer une classe Matrice qui contient les méthode **afficher()**.
  afficher() pour afficher la matrice ligne par ligne dans la console. Le remplissage des éléments de la matrice se fait de deux manières :
- La première utilise un remplissage aléatoire avec des nombres entre 0 et 9.
- La deuxième demande à l'utilisateur de remplir la matrice par exemple :

$$A = \begin{bmatrix} 1 & 2 & 0 \\ 4 & 3 & -1 \end{bmatrix}$$

  Avec n=2 lignes et m=3 colonnes et une variable booléenne qu'on appelle 'aléatoire' permet d'attribuer les méthodes de remplissage aléatoire et manuel.
- Créer les accesseurs et les mutateurs pour les variables ligne, colonne, aléatoire et matrice
- somme() qui permet de calculer la somme de deux matrices de même taille
- produitNombre() qui permet de calculer le produit d'une matrice par un nombre entier
- transposée() permet calculer la transposée d'une matrice

TP4 : Héritage et Polymorphisme

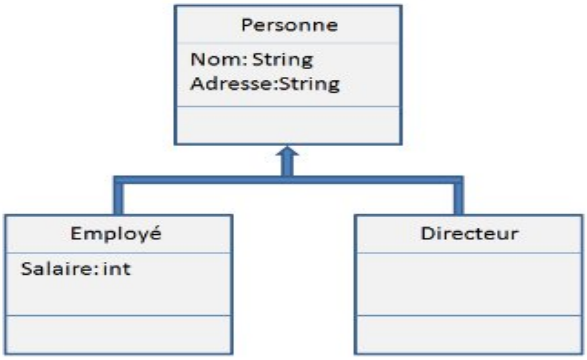
1. Introduction :

L'héritage est un mécanisme fondamental des langages de programmation par objets.

Il permet de simplifier le code d'une application et de raccourcir le temps d'écriture du code. Si une classe B hérite d'une classe A : on dit que B est une **classe fille** (ou **sous-classe**) de A et que A est une **classe mère** (ou **super-classe**) de B.

Dans ce cas, la classe fille B hérite des attributs et des méthodes de sa classe mère A. Les méthodes sont redéfinies avec le même nom, les mêmes types et le même nombre d'arguments, sinon il s'agit d'une surcharge.

Dans cet exemple, on a déclaré une classe personne avec les attributs **Nom** et **Adresse**. Les deux sous classes héritent de **Personne**: la première classe **Directeur** et la deuxième classe **Employé** qui a comme attribut supplémentaire **Salaire**.



En java, le polymorphisme est le fait que les instances d'une sous-classe, lesquelles sont substituables aux instances des classes de leur ascendance (en argument d'une méthode, lors d'affectations), gardent leurs propriétés propres. Le choix des méthodes à invoquer se fait lors de l'exécution du programme en fonction de la nature réelle des instances concernées.

2. Rappel :

a) Classes abstraites :

Au sommet d'une hiérarchie de classe, il n'est pas toujours possible de donner une définition générale de certaines méthodes, compatibles avec toutes les sous-classes, même si l'on sait que toutes ces sous-classes vont effectivement implémenter ces méthodes.
Une méthode abstraite, est incomplètement spécifiée :

- Elle n'a pas de *définition* dans la classe ou elle est introduite (pas de corps)

- Elle sert à imposer aux sous classes (non abstraites) qu'elles doivent définir la méthode abstraite héritée
- Elle doit être contenue dans une classe abstraite
- Une classe abstraite est une classe désignée comme telle au moyen du mot réservé **abstract**
- Elle ne peut être *instanciée*. Cependant, c'est possible d'implémenter le constructeur d'une classe abstraite
- Une classe abstraite doit avoir **au moins une méthode abstraite**

Exemple :

```
Class FigureFermee {
//  …

// difficile à définir à ce niveau !
Public double surface(. . .) { ??? }

// . . . pourtant la méthode suivante en aurait besoin !
Public double volume(double hauteur)  {
Return hauteur * surface() ;
}
}
```

Définir **surface** de façon arbitraire sachant qu'on va redéfinir plus tard n'est pas une bonne solution (source d'erreurs). La solution sera donc de déclarer la méthode **surface** comme **abstraite**. Il faut aussi signaler que la méthode doit exister dans chaque sous-classe sans qu'il soit nécessaire de la définir dans la super-classe.

Syntaxe : abstract Type nom_methode(liste d'arguments) ;

Exemple :

```
abstract class Personnage {
// . . .
  public abstract void afficher() ;
   //  . . .

}
```

On revient à notre exemple :

```
abstract class FigureFermee {
public abstract double surface() ;

// On peut utiliser une méthode abstraite :
Public double volume(double hauteur)  {
  return hauteur * surface() ;
  }
}
```

b) Héritage :

En Java, l'héritage est simple et est spécifié en utilisant le mot réservé **extends** en mettant public class *NomSousClasse* **extends** *NomSuperClasse* { . . .}

Exemple :

```
Public class Point3D extends Point {
    Private double z;
}
```

Si on veut qu'un attribut ou une méthode soit encapsulé pour l'extérieur mais qu'il soit accessible par les sous-classes, il faut le déclarer **protected** à la place de **private**. Le modificateur d'accès **protected** permet de rendre visible les attributs d'une classe depuis ses classes filles. La première instruction d'un constructeur d'une classe fille est toujours l'appel d'un constructeur de la classe mère en utilisant le mot- clé **super**.
Au même titre que le mot-clé **this** permet de faire référence à l'objet en cours, le mot-clé **super** permet de désigner la classe mère (superclasse), c'est-à-dire qu'à l'aide du mot-clé **super**, il est possible de manipuler les données membres et les méthodes de la superclasse.

Exemple :

```
Public Point3D (double x, double y, double z) {
    Super(x, y);  // appel  «super-constructeur»
    this.z= z ;
}
```

Il est possible de redéfinir une méthode héritée dans une classe fille. La méthode redéfinie est précédée du commentaire **@Override** dans éclipse.
Si on souhaite qu'une **méthode** ne puisse pas être redéfinie, il suffit de faire précéder son entête de mot clé **final**

Exemple :

```
@Override
Public void affichage() {
    System.out.println("("+this.x+","+this.y+","this.z+")");
}
```

c) Polymorphisme :

Le polymorphisme dans java veut simplement dire qu'une classe peut prendre plusieurs formes et c'est d'autant plus vrai avec les classes qui hérite d'une classe supérieure.
En redéfinissant une méthode dans une sous-classe, on peut spécialiser le comportement d'une méthode.
Si la classe B hérite de la classe A alors toute méthode m de A peut-être invoquée sur une instance de la classe B.

Le polymorphisme autorise la référence et l'objet à être de types différents.

```
Animal  monChien = new Chien() ;
```

Avec le polymorphisme le type de la référence peut être du type de la superclasse de l'objet.

Exemple : Il est possible de créer des tableaux polymorphes
Animal [] zoo = new Animal[3] ;
zoo[0] = new Chien() ;
zoo[1] = new Tigre() ;
zoo[2] = new Singe() ;
for (int i=0; i< zoo.length; i++) {
zoo[i].manger();
}

Grace à l'héritage et au polymorphisme, une seule et même méthode peut être utilisée pour plusieurs objets de types différents.
Dans une hiérarchie de classes :

- Un objet d'une sous-classe hérite le type de sa super-classe
- L'héritage est transitif (La relation est transitive : si A hérite de B et B hérite de C, alors A hérite de C)
- L'Héritage multiple : au moins une classe hérite d'au moins deux autres super-classes. Il s'oppose à l'héritage simple, dans lequel une classe ne peut hériter que d'une seule super-classe.
- Une classe dérivée **hérite** des attributs publics de la classe mère, donc un objet peut donc avoir **plusieurs types** selon la classe héritée.

Exemple d'application:

On souhaite créer une classe **ColoredPoint** qui représente un point coloré dans le plan. Alors on peut utiliser la classe **Point** et lui ajouter des propriétés.

```
class ColoredPoint extends Point {
    private byte color;
    public void setColor(byte c) { color = c; }
    public byte getColor() { return color; }
}

ColoredPoint p = new ColoredPoint();
p.setColor((byte)50);
System.out.println(p.toString() + " with color " + p.getColor());
```

Un objet de type **ColoredPoint** étant aussi de type **Point**, il peut donc utiliser les méthodes getX(), getY() en plus de getColor().

Exercice 1 :

- Donnez la définition d'une classe **Cercle** qui est décrit par son rayon (attribut privé de type double). La classe **Cercle** propose un constructeur **public**, **Cercle(double rayon)** qui initialise le rayon via une valeur passée en paramètre.
- Définir les deux méthodes publiques pour la classe **Cercle** : une pour retourner le rayon et une autre qui retourne son aire.
- Définir la classe **Cylindre**, sous-classe de la classe Cercle. Cette classe proposera 2 constructeurs : un avec seulement le rayon et un dernier avec le rayon et la hauteur du cylindre.

- Ajouter des méthodes pour retourner la hauteur et le volume du cylindre. Identifier les méthodes héritées de **Cercle** qui doivent être redéfinies et réécrivez-les.
- Définir la classe **Test** pour calculer le volume du cylindre avec un rayon=5 et avec un (rayon=3 et hauteur=9)

Exercice 2 :

- Réalisez le programme avec les deux classes **Employe** et **Cadre** qui hérite de la classe **Employe**. La classe **Employe** a deux variables d'instances **nom** et **salaire**. La classe **Cadre** possède un attribut supplémentaire **prime.**
- Définir deux méthodes **getNom()**, **getSalaire()** qui permettent d'accéder au nom et salaire de l'employé, et la méthode **toString()** permettant d'afficher les informations de la classe **Employe**.
- Définir la méthode **getPrime()** qui permet d'accéder à la valeur de la prime
- Redéfinir la méthode **getSalaire()** et la valeur du salaire en réintégrant la valeur de la prime dans la valeur du salaire.
- Ensuite dans la classe principale **Test**, créer un tableau dynamique contenant :
  Employe(''Jean'', 1543.) ;
  Employe(''Janine'', 1231.) ;
  Cadre(''Serge'', 2100., 235.) ;

Exercice 3 :

- Créer la classe **Animal** dans laquelle nous allons introduire la méthode **communiquer()** qui va permettre l'affichage du message suivant **: '**Un animal communique avec les autres'
- Créer les sous classes **Chat** et **Chien** qui héritent de la super classe **Animal.** Pour le méthode d'héritage **communiquer(),** on dit que le chien **aboie** mais le chat **miaule**
- Comment on appelle cette méthode d'héritage ?
- Créer la classe **Test** pour afficher la méthode **communiquer()** correspondante aux instances animal, chat et chien.

Exercice 4 :

- Créer une classe **Point** qui est caractérisé par les deux attributs d'instances (x, y) et un attribut de classe appelé **nbreDePoints** qui représente le nombre de points (en termes d'objets) crées dans l'espace. Ensuite créer une classe **Point3D** qui permet d'instancier des points de l'espace avec un attribut d'instance supplémentaire z. La classe **Point3D** hérite de la super classe **Point**.
- Créer le supe constructeur de la super classe avec les cordonnes (x, y). Ensuite le constructeur avec les cordonnes (x, y, z).
- Créer une méthode qui permet l'affichage des coordonnées des points dans le plan.
- Redéfinir la méthode d'affichage pour la classe **Point3D** dans l'espace
- Créer une méthode **getNbreDePoints()** avec les accesseurs qui permet de retourner le nombre de points crées dans l'espace.

- Créer une méthode **Test** pour test les points 3D. En première étape, créer le point avec les cordonnées (1, 2, 3) et calculer le nombre de points crées. En deuxième étape, créer un autre point avec les cordonnées (1.5, 4, 6) et recalculer le nombre de points crées.
- Créer un tableau qui contient deux points. Le premier point étant un point du plan avec (x=1, y=2) et le second est un point de l'espace avec (x=2, y=4, z=6) et afficher les deux points. Que remarquez-vous ?

TP5 : Interface Graphique et Animation

1. Introduction :

La quasi-totalité des programmes informatiques, dans notre cas Java nécessite en premier lieu l'affichage de questions posées à l'utilisateur. En deuxième lieu, l'entrée de données par l'utilisateur au moment de l'exécution et au final l'affichage d'une partie des résultats obtenus par le traitement informatique. Cet échange d'informations peut s'effectuer avec une interface utilisateur (**UI** en anglais) en mode texte (ou console) ou en mode graphique.
Une interface graphique est formée d'une ou plusieurs fenêtres qui contiennent divers composants graphiques (widgets) tels que

- Boutons
- Listes déroulantes
- Menus
- Champ texte –
- etc.

Les interfaces graphiques sont souvent appelés GUI d'après l'anglais Graphical User Interface.
L'utilisateur peut interagir à tout moment avec plusieurs objets graphiques : cliquer sur un bouton, faire un choix dans une liste déroulante ou dans un menu, remplir un champ texte, etc … Ces actions peuvent modifier totalement le cheminement du programme, sans que l'ordre d'exécution des instructions ne puisse être prévu à l'écriture du code.
Les boîtes à outils graphiques offrent des facilités pour utiliser et gérer la file d'attente des événements. En particulier pour associer les événements avec les traitements qu'ils doivent déclencher.

2. Rappel :

Dans ce TP, nous allons aborder la notion de création d'une interface graphique **JFrame** avec **swing**. La fenêtre est le panneau principal qui englobe tous les composants graphiques, elle est implémentée comme une instance de la bibliothèque graphique **swing** qui est issue de l'interface de programmation **AWT (Abstract Window Toolkit)**. Cette API est indépendante du système d'exploitation pour mettre en œuvre des composants graphiques. **swing** est composée de plusieurs packages. Chaque package contient des classes qui peuvent être exploitées pour le développement d'interfaces graphiques.
JFrame a un **contentPane** qui est le panneau principal, il englobe les autres composants: **JPanel**, **JMenu**, **JButton**, etc.

Ensuite, il faut poser des composants dans cette fenêtre (bouton, textes, …) et cela nécessite un espace d'exécution appelé **conteneur** : on utilise la classe **JPanel**. Notons qu'en programmation, un **conteneur** est une structure de données qui représente une collection d'autres données. En **Java** un **conteneur** désigne une sous-librairie fournissant un ensemble de services spécialisés, à une application logicielle.

Ensuite, on pose les composants dans le conteneur :
texte statique : classe **JLabel**
bouton : classe **JButton**
texte dynamique (champ de texte) : classe **JTextField**
image : classe **ImageIcon**

Pour que votre application soit interactive, il faut créer une interface, c'est-à-dire qu'il faut ajouter des écouteurs d'événements aux composants qui doivent réagir aux actions utilisateur. Il faut faire de même avec la souris et le clavier si on veut les '**écouter**'.
Les boutons et les champs de texte utilisent la classe **ActionListener**
La souris utilise la classe **MouseListener**.
Le clavier utilise la classe **KeyListener.**

Pour le dessin, il existe la Classe **Graphics** issue de la librairie **awt**. Pour dessiner dans un **JPanel**, vous utilisez cette classe **Graphics**.
On peut dessiner des **formes** (segments, rectangles, ellipses, …).
On peut écrire du **texte** avec des caractéristiques particulières.
On peut dessiner une **image.**
La méthode de référence pour dessiner est **paintComponent** (**Graphics g**). Cette méthode est appelée automatiquement une seule fois à la création de la fenêtre. On peut la rappeler quand on veut dans notre programme avec l'instruction **repaint()** qui peut être aussi utilisée pour redessiner la forme initiale à intervalles réguliers dans le cas d'une animation.
Pour dessiner dans une méthode quelconque, il suffit d'accéder à l'objet Graphics associé au conteneur de la fenêtre en ajoutant à l'intérieur de la méthode l'instruction **Graphics g = getGraphics() ;**

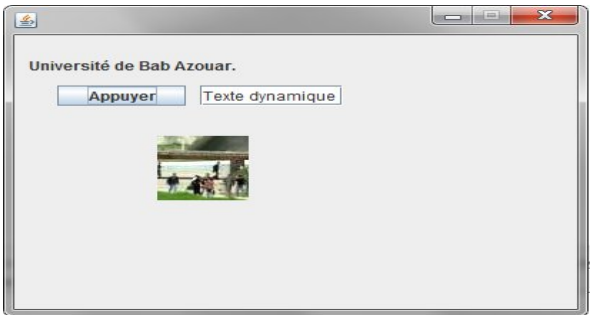
La classe utilisée pour créer un dessin animé est la Classe **Timer** de la librairie Swing. On crée un chronomètre (**Timer**) qui va nous permettre une suite d'instructions à intervalles réguliers et donc créer une illusion de mouvement.

3. Objectifs :

- Création d'une interface graphique **JFrame** avec **swing** en utilisant quelques composants (**JLabel**, **JTextField**, **JButton** et **ImageIcon**).
- Ajout des Ecouteurs d'événements à un programme pour réaliser une interface d'une application graphique
- Dessiner avec la classe **Graphics**
- Création d'une animation

Exercice 1 :

- Créer trois clases appelées : Principal, Fenetre, ConteneurFenetre
- Ensuite afficher la fenêtre suivante centrée de taille (400, 300) avec un Texte statique (Université de Bab Azouar) au dessus avec les coordonnées (10, 20, 350, 20), un bouton Appuyer (30, 50, 90, 20), un champ de texte dynamique (130, 50, 100, 20) et une image centrée sur la fenêtre positionnée à (100, 100, 64, 64). Notons que la taille est exprimée en 'points typographiques' (et non en pixels).
- Pour insérer une image il faut créer un dossier appelé 'Images' en suite il faut importer l'image (USTHB.png) dans ce dossier.



Exercice 2 :

- Créer une fenêtre qui hérite de la classe JPanel de mêmes caractéristiques que l'exercice1.
- Créer une interface pour le bouton avec l'instrution (bouton.addActionListener). Si on clique sur le bouton, le label du texte en haut change et devient "On a cliqué sur le bouton."
- Créer un écouteur d'événement pour le champ de texte dynamique avec l'instruction (champTexte.addActionListener). Lorsque on appuie sur la touche Entrée du clavier et on sera dans le champ de texte, l'étiquette va afficher le contenu qui est dans le champ de texte.
- Ajouter une interface pour la souris et afficher la position ou on clique avec la souris pour déplacer l'image (USTHB.png).
- Ajouter une interface pour les touches du clavier pour déplacer l'image dans tous les sens, application intéressante dans le cas d'un jeu vidéo.
- Si on clique sur le bouton (Mouse), est-il possible de déplacer l'image avec le clavier, expliquer pourquoi ?

Exercice 3 :

- Créer une fenêtre qui hérite de la classe JPanel de mêmes caractéristiques que l'exercice1.
- Dessiner un Rectangle vide qui a la couleur selon le système RGB (100, 200, 250) avec les coordonnées du sommet rectangle (10, 10) avec une largeur =200 et une hauteur=250.
- Dans le même rectangle, dessiner un Rectangle plein d'une autre couleur prédéfinie (verte) avec les coordonnées du sommet rectangle (20, 20) avec une largeur =180 et une hauteur=230.

- Dessiner une ellipse pleine d'une autre couleur prédéfinie (violet) avec les coordonnées du sommet de l'ellipse (40, 40) avec une largeur =100 et une hauteur=150.
- Ecrivez une chaine de caractère (Dessin) avec des caractéristiques particulières (Couleur noir, Position du texte (260, 50), Font : Arial, Style : Gras, Taille = 20).
- Afficher l'image (USTHB.png) à la position (260, 100) avec la fonction 'g.drawImage'.

Exercice 4 :

- Créer une fenêtre vide qui hérite de la classe JPanel de mêmes caractéristiques que l'exercice1.
- Créer le conteneur de la fenêtre
- Dessiner un Rectangle plein qui a la couleur bleu avec les coordonnées du sommet rectangle (10, 10) avec une largeur =20 et une hauteur=30.
- Déplacer ce Rectangle sur toute la ligne de la fenêtre horizontalement sans action extérieure en utilisant le Timer de la librairie Swing avec un temps d'attente de 10 ms
- Que pouvez-vous remarquez dans l'empilement du dessin du Rectangle

Solution TP0 :

Exercice 1 :

```
// Ce programme calcule le quotient de 2 entiers donnés et l'affiche dans
la console
public class Ex3 {   // Ne pas modifier cette ligne
      public static void main(String[] args) {
            // Déclaration des variables
            int diviseur;
           // Si on mets dividende avec le type entier (int),
l'ordinateur fait la division entière (division Euclidienne, quotient=2 et
le reste=1)
            double dividende, quotient;

            // Initialisation des variables
            dividende= 5;
            diviseur= 2;
            // Calcul du quotient
            quotient= dividende / diviseur;

            // Affichage du résultat dans la console
      System.out.print("Le quotient de la division est " + quotient);
      }
}
```

Exercice 2 :

```
// Importation de l'outil Scanner puisqu'elle n'est pas reconnue par
Eclipse
import java.util.Scanner;

// Ce programme teste si un nombre donné est Positif, Nul ou Négatif
l'affiche dans la console
public class Ex4 {  // Ne pas modifier cette ligne
      public static void main(String[] args) {
            // Déclaration des variables
            double i;
            // Enregistrement du nombre tapé au clavier par l'utilisateur
            Scanner clavier= new Scanner(System.in);
            System.out.print("Entrer la valeur du nombre : ");
            i=clavier.nextDouble();
            if (i>0)
                  System.out.print("Ce nombre est Positif ! : ");
            Else if (i==0)
                   System.out.print("Ce nombre est Nul ! : ");
            else
                  System.out.print("Ce nombre est Négatif ! : ");
            // Libération de l'espace mémoire utilisé par le Scanner
            clavier.close();
      }
}
```

Exercice 3 :

```
import java.util.Scanner;

public class Ex5 {
	//
	public static void main(String[] args) {
		// Déclaration des variables
		int codeSolde;
		double prixArticle, prixSolde;
		// Enregistrement du prix et du code tapés par l'utilisateur
		Scanner clavier = new Scanner(System.in);
	System.out.print("Entrer le prix de l'article non soldé en Da : ");
		prixArticle = clavier.nextDouble();
System.out.print("Entrer le code Solde, 1 pour -20%, 2 pour -50% ou 3 pour
-70% : ");
		codeSolde = clavier.nextInt();
		// Calcul du prix soldé et affichage
		if (codeSolde == 1) {
			prixSolde = 0.8 * prixArticle;
System.out.print("Le prix de l'article soldé a -20% est "+prixSolde+
"Da");
		} else if (codeSolde == 2) {
			prixSolde = 0.5 * prixArticle;
System.out.print("Le prix de l'article soldé a -50% est "+prixSolde+
"Da");
		} else {
			prixSolde = 0.3 * prixArticle;
System.out.print("Le prix de l'article soldé a -70% est "+prixSolde+
"Da");
		}
		// Libération de l'espace mémoire utilisé par le scanner
		clavier.close();
	}
}
```

On peut afficher des messages dans la console grâce à ces instructions :

System.out.println, qui affiche un message avec un saut de ligne à la fin

System.out.print, qui affiche un message sans saut de ligne

Exercice 4 :

```
// A refaire Ex5 avec la fonction à choix multiples
import java.util.Scanner;

public class Ex06 {
	public static void main(String[] args) {
		// Déclaration des variables
		int i, ii;

		// Enregistrement du nombre tapé par l'utilisateur
		Scanner clavier= new Scanner(System.in);
System.out.print("Entrer le code de la valeur du nombre, 1 pour un nombre
positif, 2 pour un nombre nul ou 3 pour un nombre negatif : ");
```

```
                i=clavier.nextInt();
                switch (i) {
                case 1:
System.out.print("Entrer le code de la valeur du nombre : ");
                        ii=clavier.nextInt();
                        System.out.print("Ce nombre est positif ! ");
                        break;
                case 2:
System.out.print("Entrer le code de la valeur du nombre : ");
                        ii=clavier.nextInt();
                        System.out.print("Ce nombre est nul ! ");
                        break;
                case 3:
System.out.print("Entrer le code de la valeur du nombre : ");
                        ii=clavier.nextInt();
                        System.out.print("Ce nombre est négatif ! ");
                        break;
                }

                // Libération de l'espace mémoire utilisé par le scanner
                clavier.close();
        }
}
// A refaire Ex6 avec la fonction à choix multiples
import java.util.Scanner;
public class Ex6 {
        public static void main(String[] args) {
                // Déclaration des variables
                int codeSolde;
                double prixArticle, prixSolde;
                // Enregistrement du prix et du code tapés par l'utilisateur
                Scanner clavier = new Scanner(System.in);
System.out.print("Entrer le prix de l'article non soldé en Da : ");
                prixArticle = clavier.nextDouble();
System.out.print("Entrer le code Solde, 1 pour -20%, 2 pour -50% ou 3 pour
-70% : ");
                codeSolde = clavier.nextInt();
                // Calcul du prix soldé et affichage
                switch (codeSolde) {
                case 1:
                        prixSolde = 0.8 * prixArticle;
System.out.print("Le prix de l'article soldé a -20% est " + prixSolde +
"Da");
                        break;
                case 2:
                        prixSolde = 0.5 * prixArticle;
System.out.print("Le prix de l'article soldé a -50% est " + prixSolde +
"Da");
                        break;
                case 3:
                        prixSolde = 0.3 * prixArticle;
System.out.print("Le prix de l'article soldé a -70% est " + prixSolde +
"Da");
                        break;
                }
                // Libération de l'espace mémoire utilisé par le scanner
                clavier.close();
        }
}
```

Solution TP1 :

Exercice 1 :

Le rôle du programme est de calculer le quotient de deux entiers tapés par l'utilisateur et l'affiche dans la console. Il permet aussi de lire une saisie dynamique d'un nombre tapé au clavier, à la fois pour dividende et diviseur via l'objet clavier de la classe 'Scanner' avec l'entrée standard, 'System.in'.

Exercice 2 :

Voiture.Java

```
public class Avion {
        // Attributs de la classe
        public int id;
        public String marque;
        public double vitesse;
        public int puissance;

        // Variable de l'objet
        // Variables dont il n'existe qu'un seul exemplaire
        // associé à sa classe de définition
// Variable pas spécifique à un objet mais spécifique à une classe, elle
ne sera pas une variable d'instance, tous les objets vont partager cette
variable
        public static int count = 0;
        /* Les variables statiques n'existent qu'en un seul exemplaire pour
toutes les instances de la classe, au contraire des variables objet.
        Elles sont utilisables sans avoir besoin d'instancier la classe */
// Création du constructeur avec les différents attributs marque, vitesse,
        // puissance
        public Avion(String marque, double vitesse, int puissance) {
                this.id = ++count;
                this.marque = marque;
                this.vitesse = vitesse;
                this.puissance = puissance;    }
// 1ere méthode d'affichage de type void permettant d'afficher les
informations
public void affichage() {
                System.out.println("id=" + id + ", marque=" + marque + ",
vitesse=" + vitesse + " km/h , puissance=" + puissance);

// 2eme méthode d'affichage toString ( ) de type String permettant
d'afficher les informations
Instruction return à l'intérieur de la méthode. C'est elle qui effectue le
renvoi de la valeur, ici un string
public String toString() {
                return "id=" + id + ", marque=" + marque + ", vitesse=" +
vitesse + "km/h , puissance=" + puissance;
        }
}
```

Puis compiler le programme principal : **TestAvion.Java**

```
public class TestAvion {
```

```
        // Programme principal
        public static void main(String[] args) {
                Avion Boeing = new Avion("Boeing", 920, 3000);
                Avion ATR = new Avion("ATR", 554, 2750);
                Voiture Airbus = new Avion("Airbus", 1020, 3700);
// 1ère méthode d'affichage des informations de l'avion
                Boeing.affichage();
                ATR.affichage();
                Airbus.affichage();
// 2ème méthode d'affichage des informations de l'avion
                System.out.println(Boeing);
                System.out.println(ATR);
                System.out.println(Airbus);

// Pour y accéder à une variable de classe, il faut utiliser non pas un
identificateur d'objet mais le nom de la classe
System.out.println("Le nombre d'avions est : " + Avion.count);
        }
}
```

Exercice 3 :

Tirelire.java

```
/* Création d'une classe Java nommée Tirelire de visibilité public,
 *  ayant pour attributs : public Double montant */

public class Tirelire {
        // Attribut de l'objet
        public Double montant;

        /* Création d'un constructeur ayant comme paramètres montant */
        public Tirelire(Double s) {
                this.montant = s;
        }

        /* Création d'une méthode deposer() du type void qui gère les
versements */
        public void deposer(Double d) {
                this.montant += d;
        }

        /* Création d'une méthode retirer() du type void qui gère les
retraits */
        public void retirer(Double r) {
                this.montant -= r;
        }

        /*
         * Création d'une méthode afficher() du type void permettant
d'afficher le montant
        public void afficher() {
                System.out.println("Votre montant est de : " + this.montant
                                + " Dinars");
        }
}
```

Puis compiler le programme principal : **TestTirelire.java**

```
/* Création d'une classe TestTirelire permettant de voir le montant de la
Tirelire en effectuant un versement et puis un retrait */

public class TestTirelire {
	// Programme principal
	public static void main(String[] args) {

		Tirelire monMontant = new Tirelire(5000.0);
		monMontant.deposer(2000.0);
		monMontant.retirer(1000.0);
		monMontant.afficher();
	}
}
```

Exercice 4

Quadrilatère.java

```
// Création d'une classe Java nommée Quadrilatère ayant pour attributs :
Longueur et Largeur tous du type Double, ajouter ensuite un constructeur
avec paramètres.
public class Quadrilatère {
	// Attributs de l'objet
	public Double Longueur;
	public Double Largeur;

	/* Création d'un constructeur ayant comme paramètres Longueur et
Largeur */
	// Le constructeur d'un objet porte le même nom que la classe et ne
possède aucune valeur de retour
	public Quadrilatère(Double L, Double l) {
		this.Longueur = L;
		this.Largeur = l;
	}

	// La définition d'une méthode s'appelle "déclaration".
	// Définition d'une méthode surface() du type Double permettant de
calculer
	// la Surface du Quadrilatère
	// Les arguments sont facultatifs, mais s'il n'y a pas d'arguments,
les parenthèses doivent rester présentes
	// la surface du Quadrilatère
	public double surface() {
// La méthode peut renvoyer une valeur (et donc se terminer) grâce au
// mot-clé return.
		return this.Longueur * this.Largeur;
	}

// Définition d'une méthode périmètre() du type Double permettant de
	// calculer le périmètre du Quadrilatère
	public Double perimetre() {
		return 2 * (this.Longueur + this.Largeur);
	}

// Création d'une méthode afficher() du type void pour afficher les
```

```
        // résultats
        public void afficher() {
                System.out.println("La surface du quadrilatère est : " +
this.surface());
                System.out.println("Le périmètre du quadrilatère est : " +
this.perimetre());
        }
}
```

- Un objet est une instance de classe, la création d'objets s'appelle donc l'instanciation. Cette instanciation se fait grâce à l'opérateur new suivi du nom de la classe à instancier et de parenthèses contenant les paramètres d'instanciation (parenthèses vides s'il n'y a pas de paramètres). En réalité l'opérateur new, lorsqu'il est utilisé, fait appel à une méthode spéciale de la classe: le constructeur. Cette étape d'instanciation se fait au niveau de la création de l'objet **monQuadrilatère**.

Solution TP2 :

Exercice 1 :

Personne.java

```
import java.util.Scanner;
public class Personne {

//Déclaration des variables
private String Nom;
private String Prénom;
private int Age;
private String Job;

public Personne(){
	//Création du constructeur
}

public void inscription() { //Création d'une méthode qui permet d'inscrire
les informations d'une personne
	System.out.println("Début de l'inscription");
	Scanner sc = new Scanner(System.in);
	System.out.print("Votre nom :");
	this.Nom = sc.nextLine();
	System.out.print("Votre prénom :");
	this.Prénom = sc.nextLine();
	System.out.print("Votre job :");
	this.Job = sc.nextLine();
	System.out.print("Votre age :");
	this.Age = sc.nextInt();
}
public void afficher() { //Création d'une méthode qui permet d'afficher
les informations de la personne
		System.out.println(" Nom : "+this.Nom+" Prénom :
"+this.Prénom+" Age : "+this.Age+" Travail : "+this.Job);
}

//Création des Setters et Getters pour les différents attributs

public int getAge() {
	return Age;
}

// Affichage du message suivant ('Accès interdit aux mineurs !') si l'âge
inséré est inferieur à 18 ans
public void setAge(int age) {
	if (age<18) {
		System.out.print("Accès interdit aux mineurs ! ");
	}
	else {
		Age = age;
		System.out.println("Bienvenue !");
		System.out.println(" Nom : "+this.Nom+" Prénom :
"+this.Prénom+" Age : "+this.Age+" Travail : "+this.Job);
	}
}
}
```

Puis compiler le programme principal : **TestPersonne.java**

```
public class TestPersonne {
	public static void main(String[] args) {
		Personne p1=new Personne(); //Appel à l'objet p1 de la classe Personne
		p1.inscription(); //inscription
		p1.afficher();

		// Changement de l'âge de l'étudiant avec les Setters pour vérifier la condition de l'âge majeur
		p1.setAge(15); //Modification de l'âge;

	}
}
```

Exercice 4 :

Point.java

```
import java.util.Scanner;

public class Point {
	private double x1; //Déclaration des attributs
	private double y2;
	private double x2;
	private double y1;
	private double perimetre;
	private double surface;
	private double d;
	Scanner sc=new Scanner(System.in);
	public Point() { //Création du constructeur
System.out.print("Entrez la valeur de x pour votre premier point A: ");
		x1=sc.nextDouble();
System.out.print("Entrez la valeur de y pour votre premier point A: ");
		y1=sc.nextDouble();
System.out.print("Entrez la valeur de x pour votre deuxième point B: ");
		x2=sc.nextDouble();
System.out.print("Entrez la valeur de y pour votre deuxième point B: ");
		y2=sc.nextDouble();
	}
	public double d() { //Création de la méthode pour le calcul de la distance
		return d=Math.sqrt(Math.pow(x1-x2,2)+Math.pow(y1-y2,2));
	}
	public void affichage() { //Affichage de la distance
	System.out.println("La distance entre les deux points : "+this.d);
	}
	// Création des Getters & Setters
	public double getX1() {
		return x1;
	}
	public void setX1(double x1) {
		this.x1 = x1;
	}
	public double getY2() {
		return y2;
```

```
        }
        public void setY2(double y2) {
                this.y2 = y2;
        }
        public double getX2() {
                return x2;
        }
        public void setX2(double x2) {
                this.x2 = x2;
        }
        public double getY1() {
                return y1;
        }
        public void setY1(double y1) {
                this.y1 = y1;
        }

        public double getPerimetre() { //Calcul du périmètre
                return this.perimetre= 2*this.d*Math.PI;
        }

        public double getSurface() { //Calcul de la surface
                return this.surface= Math.PI*Math.pow(this.d,2);
        }
}
```

Puis compiler le programme principal : **TestPoint.java**

```
public class TestPoint {
        public static void main(String[] args) {
                        Point p=new Point(); //Création de l'objet
                        p.d(); //Calcul de la distance
                        p.afficher(); //Affichage de la distance
System.out.println("Le périmètre de ce cercle est égale à:
"+p.getPerimetre()); //Calcul et affichage du périmètre
System.out.println("La surface de ce cercle est égale à:"+p.getSurface());
         //Calcul et affichage de la surface
        //Recalcul du périmètre et la surface avec le nouveau point
        p.setX1(2); // Donner la valeur saisie à la variable X1=2
        p.setY1(2); // Donner la valeur saisie à la variable Y1=2
        p.d(); //Calcul de la distance
        p.affichage(); //Affichage de la distance
}
}
```

Exercice 5 :

Point.java

```
package Ex5;
// Création de la classe Point
public class Point {

        // Déclaration des attributs x et y avec les accesseurs
```

```
        private int x;

        public int getX() {
                return x;
        }
        private int y;
        public int getY() {
                return y;
        }

        // Un constructeur qui initialise les valeurs de x et y
        // Le mot clé "This" différencie les attributs et les paramètres que
reçoit
        // le constructeur
        public Point(int x, int y) {
                this.x = x;
                this.y = y;
        }

        // Une méthode toString() pour afficher les coordonnées du point
        public String toString() {
                return "Point(" + x + "," + y + ")";
        }

// Une methode qui calcule la distance entre le centre et un point en
        // utilisant la classe Math
        public double distance(Point p) {
                int a = p.x - this.x;
                int b = p.y - this.y;
                return (Math.sqrt(a * a + b * b));
        }
}
```

Cercle.java

```
package Ex5;
//Création de la classe Cercle
public class Cercle {

        // Déclaration des attributs centre et rayon
        private Point centre;
        private double rayon;

        // Création de la classe Cercle en précisant son centre et son rayon
        public Cercle(Point c, double r) {
                this.centre = c;
                this.rayon = r;
        }

// Création de la classe Cercle en précisant son centre et un point du
        // cercle
        public Cercle(Point c, Point p) {
                this.centre = c;
                this.rayon = centre.distance(p);
        }

        // getSurface() : retourne le périmètre du cercle
```

```
        public double getSurface() {
                return Math.PI * Math.pow(rayon, 2);
        }

        // getPerimetre() : retourne le périmètre du cercle
        public double getPerimetre() {
                return 2 * Math.PI * rayon;
        }

        // appartient(Point p) : retourne si le point p appartient ou non à
        // l'intérieur du cercle
        public boolean appartient(Point p) {
                double d = centre.distance(p);
                return (d <= rayon);
        }

// toString() : retourne une chaine de caractères de type CERCLE(x,y,R)
        public String toString() {
return ("Cercle(" + centre.getX() + "," + centre.getY() + "," + rayon +
")");
        }

}
```

Puis compiler le programme principal : **Test.java**

```
package Ex5;

// Tester le programme
public class Test {

        public static void main(String[] args) {
                // Création de la classe Cercle en précisant son centre
c(130, 100) et
                // son rayon r=40
                Point p = new Point(130, 100);
                Cercle c1 = new Cercle(p, 40);

// Création de la classe Cercle en précisant son centre c(100, 100) et
                // un point p(200, 200)
                Point p1 = new Point(100, 100);
                Point p2 = new Point(200, 200);
                Cercle c2 = new Cercle(p1, p2);

// Affichage du périmètre et de la surface des deux cercles
                // Cercle1
System.out.println("Perimetre du " + c1.toString() + " est :"
                                + c1.getPerimetre());
System.out.println("Surface du " + c1.toString() + " est :"
                                + c1.getSurface());
                // Cercle2
System.out.println("Perimetre du " + c2.toString() + " est :"
                                + c2.getPerimetre());
System.out.println("Surface du " + c2.toString() + " est :"
                                + c2.getSurface());

// Affichage si le point p3(120, 100) appartient à l'intersection des
```

```
            // deux cercles
            Point p3 = new Point(120, 100);
      if ((c1.appartient(p3)) && (c2.appartient(p3)))
      System.out.println(p3 + " appartient a l'intersection du " + c1
                                 + "et " + c2);
            else
System.out.println(p3 + " n'appartient pas a l'intersection du "
                                 + c1 + "et " + c2);
      }
}
```

Solution TP3 :

Exercice 1 :

Avion.Java

```
public class Avion {

	// Déclaration des attributs
	public int id;
	public String marque;
	public double vitesse;
	public int puissance;
	public static int count;

	// Déclaration du constructeur
	public Avion(String m, double v, int p) {
		this.id = ++count;
		this.marque = m;
		this.vitesse = v;
		this.puissance = p;
	}

	// Affichage des informations de voiture
	public String toString() {
		return "id=" + id + ", marque=" + marque + ", vitesse=" +
vitesse
				+" km/h , puissance=" + puissance;
	}
}
```

Puis compiler le programme principal : **TestAvion.Java**

```
public class TestAvion {

	public static void main(String[] args) {
		// Déclaration des trois tableaux Marque, Vitesse et
Puissance
		String Tab1[] = { "Boeing", "ATR", "Airbus" };
		double Tab2[] = { 920, 554, 1020 };
		int Tab3[] = { 3000, 2750, 3700 };

		// Affichage du code d'exécution
		for (int i = 0; i < Tab1.length; i++) {
			Avion monAvion = new Avion(Tab1[i], Tab2[i], Tab3[i]);
			System.out.println(monAvion.toString());
		}
	System.out.println("Le nombre d'avions est : " + Avion.count);

	}
}
```

Exercice 2 :

Fascicule.Java

```
public class Fascicule {
	//Définir les attributs:
	public int id;
	public String titre;
	public String auteur;
	public double prix;
	public static int count;

	//Définir de constructeur:
	public Fascicule(String titre, String auteur, double prix) {
		this.id=++count;
		this.titre=titre;
		this.auteur=auteur;
		this.prix=prix;
}
	// Méthode d'affichage
			public String toString() {
return "Le prix du Fascicule " + this.titre + " de l'auteur nommé "
	+ this.auteur + " est : " + this.prix + " Da";
	}
	}
```

Puis compiler le programme principal : **TestFascicule.java**

```
import java.util.Scanner;
public class TestFascicule {
	public static void main(String[] args) {
		// Création du tableau livres avec une dimension =3
		Fascicule[] fascicules = new Fascicule[3];
		Scanner sc = null;
//Insertion des informations des trois fascicules en balayant le tableau
		for (int i = 0; i < fascicules.length; i++) {
			sc = new Scanner(System.in);
System.out.print("Donner le titre du fascicule n°" + (i + 1) + " :");
			String titre = sc.nextLine();
System.out.print("Donner le nom de l'auteur du fascicule n°" + (i + 1) + "
:");
			String auteur = sc.nextLine();
System.out.print("Donner le prix du fascicule n°" + (i + 1) + " :");
			int prix = sc.nextInt();
			fascicules[i] = new Fascicule(titre, auteur, prix);
		}

		// Affichage de l'exemple d'exécution
		for (Fascicule l : fascicules) {
			System.out.println(l);
	}
System.out.println("Le nombre de fascicules est " + fascicules.length);
		// Libération de l'espace mémoire utilisé par le scanner sc
		sc.close();
	}
 }
```

Exercice 4 :

```
// Contrôle de la saisie de température entre -20 et 40 et calcul de sa
fréquence
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.Scanner;

public class Temperature {

	public static void main(String[] args) {
		System.out.print("Introduire le nombre de températures :");
		Scanner nombre = new Scanner(System.in);
		int n = nombre.nextInt();
		ArrayList Tab = new ArrayList();
		for (int x = 1; x <= n; x++) {
			System.out.print("Introduire une température entre -20°
et 40° :");
			Scanner temp = new Scanner(System.in);
			int t = temp.nextInt();
			if (t >= -20 && t <= 40) {
				Tab.add(t);
			} else if (t < 20) {
System.out.print("Cette température est inférieur à -20° réessayer SVP
!");
				Scanner temp1 = new Scanner(System.in);
				int t1 = temp1.nextInt();
				if (t1 >= -20 && t1 <= 40)
					Tab.add(t1);
			} else if (t > 40) {
System.out.print("Cette température est supérieur à 40° réessayer SVP
!");
				Scanner temp2 = new Scanner(System.in);
				int t2 = temp2.nextInt();
				if (t2 >= -20 && t2 <= 40)
					Tab.add(t2);
			}
		}
		for (int y = -20; y <= 40; y++) {
			if (Collections.frequency(Tab, y) != 0) {
System.out.println("La fréquence de la température " + y + " est : " +
Collections.frequency(Tab, y));
			}
		}
	}
}
```

Exercice 5 :

Corpulence.Java

```
package Ex5;

public class Corpulence {

	// Variables d'instance
```

```
        private String nom;
        private int taille;
        private int poids;

        // Constructeur d'initialisation avec les trois parametres
        public Corpulence(String pNom, int pTaille, int pPoids) {
                nom = pNom;
                taille = pTaille;
                poids = pPoids;
        }

        // Methode qui calcule l'imc de l'objet courant
        private int calculImc() {
                int imc;
                if (poids <= 0 || taille <= 0) {
                        imc = 0;
                } else {
                imc = (int) (10000 * poids / (double) (taille * taille));
                }
                return imc;
        }

        // Methode qui analyse l'imc de l'objet courant
        public String analyseImc() {
                int imc = this.calculImc();
                String str = new String();

                if (imc < 16) {
                        str = "Anorexie";
                } else if (imc >= 16 && imc < 20) {
                        str = "Maigreur";
                } else if (imc >= 20 && imc < 26) {
                        str = "Normal";
                } else if (imc >= 26 && imc < 30) {
                        str = "Surpoids";
                } else {
                        str = "obésité";
                }
                return str;
        }

// Renvoie une chaine de caractères nom, taille, poids, et la valeur
donnée
        // par la méthode analyseImc
        public String toString() {
        return "Corpulence [nom=" + nom + ", taille=" + taille + ", poids="
                                + poids + ", analyseImc()=" + analyseImc() + "]";
        }

        // Compare si deux objets de classe Corpulence ont les mêmes champs
(nom,
        // taille, poids)
        public boolean equals(Corpulence objet) {
        if (objet.nom == this.nom && objet.taille == this.taille
                                && objet.poids == this.poids)
                        return true;
                else {
                        return false;
```

```
            }
        }

}
```

Puis compiler le programme principal : **Test.java**

```
package Ex5;
// On importe le ArrayList puisqu'il n est pas fournit dans le package par
defaut
import java.util.ArrayList;
public class Test {
        public static void main(String[] args) {

// Création d'un ArrayList de l'objet Corpulence qui s'appelle patient
// dans le quel on met trois enregistrements
            ArrayList<Corpulence> patients = new ArrayList<Corpulence>();

            // Enregitrement 1
            Corpulence patient01 = new Corpulence("Mohamed", 180, 75);
// On met l'enregistrement dans un tableau ArrayList avec la commande add
            patients.add(patient01);

// Enregistrement 2
            Corpulence patient02 = new Corpulence("Amine", 170, 80);
            patients.add(patient02);

// Enregistrement 3
            Corpulence patient03 = new Corpulence("Siham", 160, 45);
            patients.add(patient03);

        // Parcourir le tableau de la première case 0 jusqu'à la longueur du
            // tableau patients.size()
            for (int i = 0; i < patients.size(); i++) {
// On va aller dans le ArrayList patient et on va récupérer à l'aide
// de get l'enregistrement à la position (i)
        // Et on suite à cet enregistrement de type Corpulence on va lui
            // appliquer la méthode toString pour afficher son contenu
            System.out.println(patients.get(i).toString() + "\n");
            }

        // Création d'un autre enregistrement de l'objet patient04 avec un
contenu identique par rapport le patient02
            Corpulence patient04= new Corpulence("Amine", 170, 80);
        for (int i=0; i<patients.size(); i++) {
            if (patient04.equals(patients.get(i))== true) {
// Suppression d'un élément du tableau si les champs du patient04 sont les
mêmes que les champs du patient (i)
                patients.remove(i);
                            }
}

                    for (int i=0; i<patients.size(); i++) {
                System.out.println(patients.get(i).toString() + "\n");
                }

        }
}
```

Solution TP4 :

Exercice 1 :

Il faut utiliser les classes **Cercle.java** et **Cylindre.java**
Il faut compiler le programme principal avec : **Test.java**

Cercle.java

```
package EX1;

public class Cercle {

	private double rayon;

	// Création du constructeur sans parametre
	public Cercle() {
		rayon = 1;
	}

	// Création du constructeur avec parametre
	public Cercle(double rayon) {
		this.rayon = rayon;
	}
	// Création de la méthode qui retourne le rayon

    public double getRayon() {
		return rayon;
	}

	// Création de la méthode qui calcule la surface d'un cercle
	public double aire() {
		return Math.PI * Math.sqrt(getRayon());

	}

}
```

Cylindre.java

```
package EX1;

public class Cylindre extends Cercle {
	private double hauteur;

	public Cylindre() {
		super();
	}

	public Cylindre(double rayon) {
		super(rayon);
	}

	public Cylindre(double rayon, double hauteur) {
		super(rayon);
		this.hauteur = hauteur;
```

```
        }

        public double gethauteur() {
                return hauteur;
        }
         public double volume() {
                return aire() * gethauteur();
            }
        }
```

Test.java

```
package EX1;

public class Test {

        public static void main(String[] args) {
                Cylindre cylindre1 = new Cylindre();
                Cylindre cylindre2 = new Cylindre(5);
                Cylindre cylindre3 = new Cylindre(3, 9);

        System.out.println("Cylindre 1 volume  : "+ cylindre1.volume());

        System.out.println("Cylindre 2 volume  : "+ cylindre2.volume());

        System.out.println("Cylindre 3 volume  : "+ cylindre3.volume());

        }

}
```

Exercice 3 :

Il faut utiliser les classes **Animal.java, Chien.java** et **Chat.java**
Il faut compiler le programme principal avec : **Test.java**

Animal.java

```
package EX3;

public class Animal {
public Animal(){

}

public void communiquer(){
        System.out.println("Un animal communique avec les autres ");
}
}
```

Chien.java

```
package EX3;

public class Chien extends Animal {

	public Chien(){
	}
	@Override
		public void communiquer(){
			System.out.println("Un chien aboie");
		}
		}
```

Chat.java

```
package EX3;

public class Chat extends Animal {
		public Chat(){
		}

	@Override
		public void communiquer(){
			System.out.println("Un chat miaule");
		}
		}
```

Test.java

```
package EX3;

public class Test {

	public static void main(String[] args) {
		Animal animal= new Animal();
		Chien chien= new Chien();
		Chat chat= new Chat();

		animal.communiquer();
		chien.communiquer();
		chat.communiquer();

			}
}
```

- Il **est** alors possible **d**'appeler la **méthode d**'un objet sans se soucier **de** son type intrinsèque : il s'agit du '**Polymorphisme d'héritage'**.

Exercice 4 :
Il faut utiliser les classes **Point.java** et **Point3D.java**
Il faut compiler le programme principal avec : **Test.java**

Point.java

```
package EX4;

public class Point {
```

```
        protected double x;
        protected double y;
        private static int nbreDePoints;

        public Point(double x, double y){
                this.x=x;
                this.y=y;
                nbreDePoints++;
        }

        public static int getNbreDePoints() {
                return nbreDePoints;
        }

        // Méthode d'affichage
        public void affichage(){

        System.out.println("("+this.x+" , "+this.y+")");
        }
        }
```

Point3D.java

```
package EX4;
                public class Point3D extends Point {

                private double z ;

                public Point3D(double x, double y, double z){
            super(x, y);
            this.z=z;
                }

                @Override
                public void affichage(){
        System.out.println("("+super.x+" , "+super.y+", "+this.z+")");
                }
                }
```

Test.java

```
package EX4;

public class Test {

        public static void main(String[] args) {
                Point3D p= new Point3D(1,2,3);
                Point3D p1= new Point3D(1.5,4,6);

                p.affichage();
                p1.affichage();
System.out.println("Nombre de points 3D crées : "+
Point.getNbreDePoints());

                // Polymorphisme
                Point[] t=new Point[2];
                t[0]= new Point(1,2);
```

```
            t[1]= new Point3D(2,4,6);
            for (int i=0;i<2; i++)
                  t[i].affichage();

      }
      }
```

- On remarque que la méthode affichage permet d'afficher le premier point du tableau en utilisant la méthode affichage() de la classe Point et pour le second point dans un espace on fait appelle à la méthode d'affichage que contient la classe Point3D. C'est la notion du '**Polymorphisme'**

Solution TP5 :

Exercice 1 :

Il faut compiler le programme principal : **Principal.java**

```
package Ex1.src;
public class Principal {

	public static void main(String[] args) {
// Création d'un objet fenetrePrincipale de la classe Fenetre
Fenetre fenetrePrincipale= new Fenetre();

// La fenêtre crée par défaut est invisible
fenetrePrincipale.setVisible(true);

	}
}
```

Fenetre.java

```
package Ex1.src;
import javax.swing.JFrame;
// Cette classe Fenetre hérite de la classe JFrame
//Soulignement rouge dans JFrame, il faut importer la classe JFrame qui
n'est pas fournie dans le package par défaut
public class Fenetre extends JFrame {
	// Associer notre conteneur qu
	//on appelle pan à notre fenêtre
	ConteneurFenetre pan;
	// Creation du constructeur par défaut
	public Fenetre(){
		// Cette fenêtre hérite de la classe JFrame, il fallait
commencer par faire appel au constructeur de la classe JFrame
		super();
		// Appel constructeur de la super classe
		proprietesFenetre();

	}
		private void proprietesFenetre(){
		// Fixer la taille de la fenêtre (Largeur, Hauteur)
		this.setSize(400, 300);
		// Redimensionnement de la fenêtre impossible
		this.setResizable(false);
		// Centrer la fenêtre a l'écran au premier plan
		this.setLocationRelativeTo(null);
		// Fin du programme en fermant la fenêtre avec la croix
		this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		// Cette instruction permet à la fenêtre de rester au dessus
de toutes les fenetres ouvertes même en cliquant a cote
		this.setAlwaysOnTop(true);

		//On instancie notre panneau pour allouer l'espace mémoire
dont il a besoin
		pan = new ConteneurFenetre();
		// Intégration du conteneur panneau (pan) à la fenêtre
		this.setContentPane(pan);
```

```
        }
}
```

ConteneurFenetre.java

```
package Ex1.src;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JTextField;
// Importation de la classe JPanel
import javax.swing.JPanel;

// Ce conteneur fenetre hérite de la classe JPnael
public class ConteneurFenetre extends JPanel {
        // On va placer des composants dans cette fenetre
        private JLabel etiquette; // Texte statique
        // On place dans notre conteneur un bouton
        private JButton bouton;
        private JTextField champTexte; //On crée un champ de texte dynamique
private JLabel imageLbl;  // Conteneur de l'image pour la classe ImageIcon
        private ImageIcon iconPhoto;
        // Creation du constructeur
        public ConteneurFenetre(){
        // Appel à la super classe JPanel avec super (Facultative)
        super();
        this.proprietesConteneur();
        }

        private void proprietesConteneur(){
                // On n'utilise pas le gestionnaire de disposition mais qu'on
place tout à la main dans notre fenêtre (positionnement absolu)
                this.setLayout(null);
                this.proprietesEtiquette();
                // création d'une méthode qui affiche ce bouton
                this.propBouton();
        this.propChampTexte(); //Créer une méthode pour afficher le texte
        this.affichImage(); // Créer une méthode pour gérer l'affichage de
l'image
        }

        // Création d'une méthode qui gère les propriétés de notre étiquette
        private void proprietesEtiquette(){
                // On utilise le constructeur pour allouer l'espace mémoire
nécessaire pour notre étiquette
                etiquette = new JLabel();

                // Position et dimension de l'étiquette dans le conteneur
                // (10, 20) les coordonnées du point gauche de l'étiquette,
Largeur=350, Hauteur=20
                this.etiquette.setBounds(10, 20, 350, 20);
                // Mettre du texte de l'étiquette
                this.etiquette.setText("Université de Bab Azouar.");
                // Pour rendre l'étiquette visible sur notre fenêtre, on
l'associe au conteneur
                this.add(etiquette);

        }
```

```
// Méthode qui va créer ce bouton
        private void propBouton(){
                // Instanciation du bouton
                this.bouton = new JButton();
                // Mettre du texte sur mon bouton (Appuyer)
                this.bouton.setText("Appuyer");
                //Placer mon bouton dans la fenêtre sur la position(30, 50)
                this.bouton.setBounds(30, 50, 90, 20);
        // On associe ce bouton à notre conteneur avec l'instruction add
                this.add(bouton);
        }

        // Méthode qui va créer le champ de texte
        private void propChampTexte(){
                // Etapes similaires que dans la méthode propBouton()
                champTexte= new JTextField();
                this.champTexte.setBounds(130, 50, 100, 20);
                this.add(champTexte);
        }
        // Méthode qui va gérer l'affichage de l'image
        private void affichImage(){
                // créer un dossier Image dans le dossier du projet Java puis
y mettre une image
                imageLbl= new JLabel();
                iconPhoto= new ImageIcon("Images/USTHB.png");
                this.imageLbl.setBounds(100, 100, 64, 64); //(100,100) c'est
la position et 64*64 correspond à la taille de l'image
                this.imageLbl.setIcon(iconPhoto); // on intègre l'image à son
conteneur
                this.add(imageLbl);
        }
}
```

Exercice 2 :

Il faut utiliser les classes **Principal.java** et **Fenetre.java** de (l'exercice 1)
Il faut compiler le programme principal avec : **Principal.java**

ConteneurFenetre.java

```
package Ex2.src;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;

import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JTextField;
// Importation de la classe JPanel
import javax.swing.JPanel;


// Ce conteneur fenêtre hérite de la classe JPnael
```

```
public class ConteneurFenetre extends JPanel implements ActionListener,
MouseListener, KeyListener {
// On va placer des composants dans cette fenêtre
        private JLabel etiquette;
        // On place dans notre conteneur un bouton
        private JButton bouton;
        private JTextField champTexte;
        private JLabel imageLbl;  // Conteneur de l'image
        private ImageIcon iconPhoto;
        // Creation du constructeur
        public ConteneurFenetre(){
        // Appel à la super classe JPanel avec super (Facultative)
        super();
        this.proprietesConteneur();
        }
private void proprietesConteneur(){
// On n'utilise pas le gestionnaire de disposition mais qu'on place tout a
la main (positionnement absolu)
                this.setLayout(null);
                this.proprietesEtiquette();
                // création d'une méthode qui affiche ce bouton
                this.propBouton();
                this.propChampTexte();
                this.affichImage();
        }

// Création d'une méthode qui gère les propriétés de notre étiquette
        private void proprietesEtiquette(){
// On utilise le constructeur pour allouer l'espace mémoire nécessaire
pour notre étiquette
                etiquette = new JLabel();

                // Position et dimension de l'étiquette dans le conteneur
                // (10, 20) les coordonnées du point gauche de l'étiquette,
Largeur=350, Hauteur=20
                this.etiquette.setBounds(10, 20, 350, 20);
                // Mettre du texte de l'étiquette
                this.etiquette.setText("Université de Bab Azouar.");
                // Pour rendre l'étiquette visible sur notre fenêtre, on
l'associe au conteneur
                this.add(etiquette);
                this.addMouseListener(this);
                this.addKeyListener(this);
// Pour que le paneau clavier fonctionne pour déplacer l'image dans tous
les sens il faut lui donner le focus avec les 2 instructions suivantes
(L'oreille prioritaire à l'écoute de l'événement)
                this.setFocusable(true);
                this.requestFocusInWindow();

        }

        private void propBouton(){
                // Instanciation du bouton
                this.bouton = new JButton();
                // Mettre du texte sur mon bouton (Appuyer)
                this.bouton.setText("Appuyer");
                //Placer mon bouton dans la fenêtre sur la position(30, 54)
                this.bouton.setBounds(30, 50, 90, 20);
```

```
        // On associe ce bouton à notre conteneur avec l'instruction add
                this.add(bouton);
// Pour que le bouton puisse réagir aux actions utilisateur
                this.bouton.addActionListener(this);

        }
        private void propChampTexte(){
                champTexte= new JTextField();
                this.champTexte.setBounds(130, 50, 100, 20);
                this.add(champTexte);
                this.champTexte.addActionListener(this);
        }
private void affichImage(){
// créer un dossier Image dans le dossier du projet Java puis y mettre une
image
//L'action sur le champ de texte se fera avec le biais de la touche Entrée
                imageLbl= new JLabel();
                iconPhoto= new ImageIcon("Images/USTHB.png");
                this.imageLbl.setBounds(100, 100, 64, 64); //64*64 correspond
à la taille de l'image
this.imageLbl.setIcon(iconPhoto); // on intègre l'image a son conteneur
                this.add(imageLbl);
        }
// Méthode héritée de la classe ActionListener avec l'événement e
                @Override
                public void actionPerformed(ActionEvent e) {
                        if(e.getSource() == this.bouton){
                                // Le bouton est à l' origine de l'événement
                        this.etiquette.setText("On a clique sur le bouton.");
                }else if (e.getSource() == this.champTexte){
// Demander au Label étiquette d'afficher le texte qui est contenu dans le
champ de texte
                        this.etiquette.setText(this.champTexte.getText());
                }
        this.requestFocusInWindow(); // Rend le focus à ConteneurFenetre
                }

// Méthode héritée de la classe MouseListener avec l'événement e
        // Lorsque on va relâcher le bouton gauche de la souris, ca va
déclencher une action
        // (e.getX(), e.getY()) Coordonnées à l'endroit ou on a cliqué
        // (this.imageLbl.getWidth(), this.imageLbl.getHeight()) Récupérer
Largeur et hauteur pour être extensible à n'importe quelle image
                @Override
                public void mouseReleased(MouseEvent e) {
                        this.imageLbl.setBounds(e.getX(), e.getY(),
this.imageLbl.getWidth(), this.imageLbl.getHeight());
// Demander à l'étiquette d'afficher les coordonnées de la souris
                        this.etiquette.setText("Position de la souris : "+
e.getX() + ", " +e.getY() );

                }

                // Quand on relâche les touches du clavier ca va relancer une
action pour ensuite la récupérer avec l'événement e
                @Override
                public void keyReleased(KeyEvent e) {
// Test pour le mouvement vers le haut, on monte l'image de 10 selon Y
                        if (e.getKeyCode()== KeyEvent.VK_UP){
```

```
                        this.imageLbl.setBounds(this.imageLbl.getX(),
this.imageLbl.getY() - 10,  this.imageLbl.getWidth(),
this.imageLbl.getHeight());

// Test pour le mouvement vers le bas, on descend l'image de 10 selon Y
                    }else if (e.getKeyCode()== KeyEvent.VK_DOWN){
                        this.imageLbl.setBounds(this.imageLbl.getX(),
this.imageLbl.getY() + 10,  this.imageLbl.getWidth(),
this.imageLbl.getHeight());
// Test pour le mouvement vers la gauche, on décale l'image de 10 selon X
                    }else if (e.getKeyCode()== KeyEvent.VK_LEFT){
                        this.imageLbl.setBounds(this.imageLbl.getX() -
10, this.imageLbl.getY(),  this.imageLbl.getWidth(),
this.imageLbl.getHeight());
// Test pour le mouvement vers la droite, on décale l'image de 10 selon X
                }else if (e.getKeyCode()== KeyEvent.VK_RIGHT){
                    this.imageLbl.setBounds(this.imageLbl.getX() +10,
this.imageLbl.getY(),  this.imageLbl.getWidth(),
this.imageLbl.getHeight());
// Si on clique sur le bouton de la souris, on ne peut déplacer l'image,
puisque quand on appuis sur Appuyer, le bouton a récupéré le focus, donc
le panneau clavier a perdu son focus. Pour résoudre le problème afin que
le clavier garde son focus il faut rajouter dans la méthode
'actionPerformed' du bouton l'instruction this.requestFocusInWindow()
                }
                }
// Méthodes heritées inutilisées
                @Override
                public void mouseClicked(MouseEvent arg0) {
                    // TODO Auto-generated method stub
                }
// Méthodes inutilisées
                @Override
                public void mouseClicked(MouseEvent arg0) {
                    // TODO Auto-generated method stub
                }
@Override
                public void mouseExited(MouseEvent arg0) {
                    // TODO Auto-generated method stub
                }
@Override
                public void mousePressed(MouseEvent arg0) {
                    // TODO Auto-generated method stub

                }
@Override
                public void keyPressed(KeyEvent arg0) {
                    // TODO Auto-generated method stub

                }
@Override
                public void keyTyped(KeyEvent arg0) {
                    // TODO Auto-generated method stub
                }
}
```

Références bibliographiques

- A. Journaux. Support de cours du langage de programmation Java, 2014.
- C. Delannoy. Programmer en Java: Couvre Java 9. Editions Eyrolles, 2017.
- C. S. Horstmann, G. Cornell Au cœur de Java 2 : Volume I – Notions fondamentales. The Sun Microsystems Press. Java Series. CampusPress, 1999.
- F. Rossi, Initiation à la programmation Java. Université Paris-IX Dauphine, 1997.
- H. Bersini. La programmation orientée objet. Editions Eyrolles, 2013.
- H.M. Deitel, & P.J.Deitel. Comment programmer en Java. Éditions Reynald Goulet, 2002.
- J. Bloch, & A. Moussine-Pouchkine. Java efficace: guide de programmation. Vuibert, 2002.
- J. Brondeau. Introduction à la programmation objet en Java: cours et exercices. Dunod, 1999.
- J. d'Anjou, S. Fairbrother, D. Kehn, P. McCarthy, & J. Kellerman. The Java developer's guide to Eclipse. Addison-Wesley Professional, 2005.
- J. Murach. Murach's Android Programming (2nd Edition). Mike Murach & Associates, 2015.
- L.G. Picard. Initiation à la programmation orientée-objet avec le langage java. Pôle Informatique, 2014.
- P. Deitel, & H. Deitel. Java: How to program. Prentice Hall, 2011.
- P. Haggar. Practical Java: programming language guide. Addison-Wesley Professional, 2000.
- R. Sedgewick, & K. Wayne. Introduction to programming in Java: an interdisciplinary approach. Addison-Wesley Professional, 2017.
- S. Holzner. *Eclipse*. O'Reilly Media, Inc., 2004.
- V. Granet. Algorithmique et programmation en Java. Dunod, 2018.

Webographie

- Le site web de référence est : http://java.sun.com redirigé désormais sur le site d'Oracle à http://www.oracle.com/technetwork/java/index.html
- Il existe de nombreux livres et tutoriaux. Un premier tutorial est à : http://java.sun.com/docs/books/tutorial/ (redirigé chez Oracle)
- https://openclassrooms.com/fr/courses/6173501-debutez-la-programmation avec-java (dernière consultation en 2020)
- Eclipse: http://www.eclipse.org/downloads/
- NetBeans: https://netbeans.org/downloads/

View publication stats